НАЦІОНАЛЬНИЙ ТЕХНІЧНИЙ УНІВЕРСИТЕТ УКРАЇНИ
«КИЇВСЬКИЙ ПОЛІТЕХНІЧНИЙ ІНСТИТУТ
імені ІГОРЯ СІКОРСЬКОГО»
Інститут прикладного системного аналізу
Кафедра математичних методів системного аналізу

«На правах рукопису»
УДК 51-74

«До захисту допущено»
Завідувач кафедри
_____ О. Л. Тимощук
«___»_____20__ р.

Магістерська дисертація

на здобуття ступеня магістра

зі спеціальності 124 Системний аналіз

на тему: «Система мультимодального текстового пошуку»

Виконав:
студент ІІ курсу, групи КА-61м
Шевченко Тарас Петрович _____

Науковий керівник:
старший науковий співробітник
Інституту програмних систем
НАН України, к.ф.-м.н, ст.н. с.,
Ігнатенко Олексій Петрович _____

Рецензент:
професор кафедри автоматики та
управління факультету інформатики
та обчислювальної техніки
КПІ ім. Ігоря Сікорського,
д.ф.-м.н., проф.,
Дорошенко Анатолій Юхимович _____

Засвідчую, що у цій магістерській
дисертації немає запозичень з праць
інших авторів без відповідних посилань.
Студент _____

Київ
2018

Національний технічний університет України

«Київський політехнічний інститут імені Ігоря Сікорського»

Інститут прикладного системного аналізу

Кафедра математичних методів системного аналізу

Рівень вищої освіти – другий (магістерський)

Спеціальність (спеціалізація) – 124 «Системний аналіз» («Системний аналіз і управління»)

ЗАТВЕРДЖУЮ

Завідувач кафедри

_____ О. Л. Тимощук

«___»_____20__ р.

ЗАВДАННЯ
на магістерську дисертацію студенту

Шевченку Тарасу Петровичу

1. Тема дисертації «Система мультимодального текстового пошуку», науковий керівник дисертації Ігнатенко Олексій Петрович, к.ф.-м.н., с.н.с., затверджені наказом по університету від «___»_____ 20__ р. №_____

2. Термін подання студентом дисертації _____

3. Об’єктом дослідження є системи текстового пошуку.

4. Предметом дослідження системи мультимодального текстового пошуку

5. Перелік завдань, які потрібно розробити: оляд систем та використовуваних в них алгоритмів та структур, опис їхніх недоліків, алгоритмізація та узагальнення існуючих підходів, розробка системи мультимодального пошуку на основі обраної множини алгоритмів, обгрунтування практичної цінності отриманих результатів.

6. Орієнтовний перелік графічного (ілюстративного матеріалу): презентація (об’єкт, предмет, мета дослідження, порівняльний аналіз алгоритмів, вимірювання якості пошуку, висновки), лістинг коду.

7. Орієнтовний перелік публікацій: розширені тези “Efficient Search in Short Text Documents” та виступ на міжнародній конференції ICTERI 2018 та доповідь.

8. Дата видачі завдання _____

Календарний план

№ з/п	Назва етапів виконання магістерської дисертації	Термін виконання етапів магістерської дисертації	Примітка

Студент Шевченко Т. П.

Науковий керівник дисертації Ігнатенко О. П.

РЕФЕРАТ

Магістерська дисертація: 93 ст., 11 рис., 32 табл., 40 джерел та 2 додатки.

Протягом останніх двох століть людство експотенційно накопичує об’єми текстових даних. Зростання об’ємів накопичених документів ставить нові вимоги до швидкості та якості пошуку.

З 1954-го року проводяться дослідженні в області семантичного аналізу текстів. Новітні результати у даній області ведуться шляхом покращення моделі “Word2Vec”, анонсовоної компанією Google у 2013-му році.

Метою даної роботи є розробка системи мультимодального текстового пошуку. У роботі проаналізовано методи вирішення задачі обробки текстів, побудови векторного простору для врахування семантики слів, також розглянуто задачі знаходження тем документу на основі багатьох модальностей.

Об’єктом дослідження є системи текстового пошуку, предметом дослідження - системи мультимодального пошуку та їх компоненти. Деякі результати даної роботи опубліковано на міжнародній конференції “ICTERI 2018”.

Результати роботи:

- реалізовано систему мультимодального текстового пошуку;
- реалізовано методи пошуку у баготовимірних розріджених просторах;
- вдосконалено існуючі алгоритми текстового пошуку.

Результати даної роботи рекомендовано до використання у пошуку по Інтернет-магазинах, каталогах продукції, інформаційних веб-сайтах, файлових системах, користувацьких повідомленнях. Проведено вимірювання, перевірено гіпотези, що підтверджують практичну значимість результатів та можливості їхнього застосування.

При подальшому розвитку системи доцільно додати нові методи оцінки релевантності документів, а також створити більш гнучку мову для керування даними.

МУЛЬТИМОДАЛЬНИЙ ТЕКСТОВИЙ ПОШУК, ІНФОРМАЦІЙНИЙ ПОШУК, МАШИННЕ НАВЧАННЯ, ПОВНОТЕКСТОВИЙ ПОШУК,

ОКАРІ-BM25, ЕМ-АЛГОРИТМ, ЙМОВІРНІСНИЙ ВИСНОВОК, СЕМАНТИЧНИЙ АНАЛІЗ ТЕКСТІВ

ABSTRACT

Master’s thesis: 93 p., 11 fig., 32 tabl., 40 sources and 2 appendices.

Over the last two centuries, mankind increases data volumes all around the world. This process increases requirements on speed and quality of search engines.

Serious research around search engines started since 1954. In this years has been introduced the first Vector Space model. In 2013 Google announced “Word2Vec” model which is a de-facto standard in Vector Space representation of words.

The task is to devel multimodal text search system for search relevance evaluation. In this work were implemented and examined methods of Multimodal Information Retrieval compared different algorithms for spatial retrieval, frequency counting, and topic modeling.

The object is text search systems. The subject is multimodal text search systems.

Results:

- implemented multimodal text search system, which is based on asynchronous model;
- implemented disk-based spatial retrieval;
- generic implementation of document storage and retrieval.

Results of this work are applicable for as a search engine for Internet-stores, catalogs, informational websites, filesystems and everywhere, where there is need in high qulity search.

In the future it is useful to increase the number of ranking factors and extend existing language for working with data and retrieval operations.

MULTIMODAL INFORMATION RETRIEVAL, INFORMATION RETRIEVAL, MACHINE LEARNING, FULL-TEXT SEARCH, OKAPI-BM25, EM-ALGORITHMS, PROBABILISTIC INFERENCE, SEMANTIC TEXT ANALYSIS

ЗМІСТ

ПЕРЕЛІК СКОРОЧЕНЬ

BSD — Berkeley Software Distribution
CART - Classification And Regression Trees
k-NN — K-nearest Neighbors
SVM — support vector machines
ЕОМ — електронно-обчислювальна машина
SSL - Secure Sockets Layer DFR - Divergence-from-randomness model
BM - Best matching

ВСТУП

Процес зростання об’ємів текстових даних ставить нові вимоги до процесу пошуку потрібної інформації. На практиці дані мають різнорідну структуру, оскільки текстові дані мають різну природу: короткі тексти знаходяться поряд із довгими, при цьому також зберігаються метадані із інформацією про перегляди, дату створення та модифікації, також разом із текстом можуть зберігатися звукові та відеофайли, дані із розпізнаним звуком, а також будь-які дані, які були обрані для зберігання авторами системи.

Об’єктом дослідження у даній роботі є системи текстового пошук. Предметом дослідження є система мультимодального текстового пошуку.

Не дивлячись, на те, що більшість інформації зберігається в реляційних базах даних і є структурованою та зведеною до нормальних форм, виникають проблеми іншого роду, а саме ті, що у обумовлені людським фактором.

У даних спостерігаються помилки: дублювання ідентичних сутностей під різними ідентифікаторами, порушення цілісності даних, орфографічні помилки в текстовій інформації. Дані помилки чинять негативний вплив на якість пошукової видачі.

Окрім питання коректності даних, що зберігаються, виникає необхідність надання доступу до цих таких даних кінцевим користувачам у якості сервісів. Дана задача є актуальною для таких компаній як Wall-Mart, Kroger, Costco, Amazon, Walgreen, а також власників усіх Інтернет-магазинів та каталогів, що оперують значними об’ємами інформації.

В описаних вище умовах стає очевидною необхідність виявлення та виправлення таких помилок. Ключовим функціоналом для згаданих компаній є можливісті пошуку продукції серед мильйонів записів, що зберігаються у базах даних. У зв’язку з цим, виникає необхідність розробки платформ, що здіснюють побудову пошукового індексу з метою надання послуг кінцевим користувачам. Однак якість пошуку при використанні детермінованих алгоритмів залишає бажити кращого. Вимоги до розумінная семантики пошукового запиту та потреб ко- ристувача унеможливлюють розробку якісних алгоритмів з використанням традиційних підходів, що

призводить до необхідності використання методів інтелектуального аналізу даних, а також прийняття рішень на базі чисельної міри відповідності між документом та пошуковим запитом у режимі реального часу, що ставить строгі вимоги до якості застосуваних алгоритмів.

У зв’язку з цим виникає необхідність опитування експертів для можливості подальшого застосування технік аналізу даних, що довозволяють проводити навчання з учителем для побудови моделі, що здіснює прогнозування релевантності текстового пошукового запиту, використовуючи загальну модель.

РОЗДІЛ 1 Системи пошуку текстової інформації

1.1 Задача інформаційного пошуку

Інформаційний пошук - це напрям досліджень, пов’язаний із визначенням релевантності інформаційних ресурсів споміж даних, що зберігаються, у відповідності до мети. Інформаційний пошук може базуватися на повнотекстовому пошуку або на контентно-орієнтованому індексуванні. Системи автоматизованого пошуку використовують для зменшення інформаційного перевантаження. Бібліотеки використовують пошукові системи з метою надання доступу до книг, журналів. Головними напрямами для застосування інформаційного пошуку є:

а) цифрові бібліотеки;
б) рекомендаційні системи;
в) медійні системи;
г) пошук по блогах та новинах;
д) пошук зображеннь;
е) звуковий пошук;
ж) відео пошук;
з) пошукові системи;
и) загального призначення;
к) спеціалізовані;
л) пошук знань;
м) класифікація документів;
н) системи підсумовування між документами;
о) питально-відповідальні системи;
п) крос-лінгвістичний пошук.

Оскільки дана робота присвячена прогнозуванню релевантності текстових запитів користувача у пошукових систем, приведемо приклади спеціалізова- них пошукових систем:

а) біо-пошук;

б) географічний пошук;
в) пошук за хімічними структурами;
г) пошук сирцевих кодів;
д) законодавчий пошук.

Задача текстового пошуку є визначеною як порівняння запитів користувача із множиною існуючих записів. Ці записи можуть бути як не структорованим тестом, так і газетними статтями, записами щодо нерухопості або інструкці- ями. Користувацькі запити можуть бути різнорідними:

а) послідовністю речень;
б) описовою характеристикою;
в) слабо зв’язаними словами.

У даній розглядається задача прогнозування релевантності результатів текстового пошуку для документів із фіксованою структурою.

1.1.1 Apache Solr

Solr - вільне та відкрите програмне забезпечення, пошукова платформа реалізована на Java, базується на Apache Lucene. Основні можливості включають повнотекстовий пошук, підсвічування результатів, фасетний пошук, індексування в реальному часі, динамічну кластеризацію, інтеграцію з базами даних, NoSQL можливості, обробку складних форматів (наприклад, Word, PDF). Solr має можливості розподіленого пошуку та реплікації індексу, має дуже добру масштабовність і стійкість до відмов.

Solr запускається у якості окремого серверу. Використовує бібліотеку Lucene як ядро для впровадження повнотекстового пошуку та індексації. Apache Solr має REST-подібний HTTP/XML і JSON API, що дозволяє використовувати Solr програмами написаними на інших мовах програмування. Даний продукт має структуру, що дозволяє проводити розширення.

Основні функції програмного продукту:

а) повнотекстовий пошук на базі Lucene;
б) фасадна навігація;
в) підсвічування кліків;
г) підтримка тестових запитів;
д) підтримка структорованих запитів;
е) відсутність вимог до схеми зберігання данних;
ж) графічний HTML-інтерфейс для адмністрування;
з) кластеризація пошукових результатів;
и) кешування результатів;
к) підтримка автоматизованого менеджменту для кластерів із використанням ZooKeeper;
л) функціональні запити;
м) потокові запити.

Характеристики, що віпдовідають за безпеку:

а) підтримка SSL;
б) вбудовані можливості індектифікації та авторизації.

1.1.2 Elastic Search

Elasticsearch — вільне програмне забезпечення, пошуковий сервер, розроблений на базі Lucene. Надає розподілений, мультиарендний повнотекстовий пошуковий рушій з HTTP веб-інтерфейсом і підтримкою безсхемних JSON документів.

Застосування:

1. Реалізація пошуку по веб-сайту, наприклад пошук товарів в Інтернет-магазині. В цьому випадку Elasticsearch індексує каталоги товарів, та надає можливості пошуку та припущення щодо автозаповнення.
2. Зберігання журналів подій чи транзакцій, аналізування і добування даних для отримання тенденцій, статистик, висновків, аномалій. В даному випадку можна використовувати Logstash для збору,

об’єднання, аналізу даних, і потім перенаправляти ці дані в Elasticsearch для подальшого опрацювання.

3. Для розробки, наприклад, платформи по ціновому оповіщенню, що дозволяє досвіченим в цій сфері користувачам визначати правила типу «Я зацікавлений в придбанні електронного ґаджету XXX і я хочу бути оповіщеним якщо ціна у будь-якого постачальника впаде нижче XXX протягом наступного місяця». В даному випадку можна збирати ціни, індексувати їх в Elasticsearch і використовувати функцію зворотнього пошуку: зіставляти коливання цін з запитом користувача і при відповідності до запиту надсилати сповіщення.
4. Для впровадження аналітики/Бізнес-аналітики в проект, коли треба швидко досліджувати, аналізувати, візуалізувати надзвичайно великі об’єми даних (мільйони чи мільярди записів). В даному випадку доцільно використання Elasticsearch для збереження даних і Kibana для побудови користувацьких панелей відображення і візуалізації необхідних аспектів. До того ж, можна використовувати агрегаційні функції Elasticsearch для здійснення комплексної бізнес-аналітики даних.

1.2 Місце системного аналізу у задачі інформаційного пошуку

Пошукові системи для вдоволення користувачів потребують адаптації до дій користувача в режимі реального часу.

Задача інформаційного пошуку знаходиться на стику таких дисциплін, як:

а) математики;
б) інформатики;
в) лінгвістики.

Крім того, застосовуються алгоритми, які виникли на стику біології, математики та інформатики, наприклад нейронні мережі. Також варто зазначити, що для побудови системи було застосовано широкий

математичний апарат: математичний аналіз, теорія ймовірностей, методи оптимізації.

Види невизначеності:

а) критеріальна невизначеність;
б) ситуаційна невизначеність;
в) системна невизначеність;
г) ситуаційна невизначеність [1].

Ситуаційна невизначеність погягає в тому, що не можна попередньо визначити характер вхідних даних, а також якість мережевого з'єднання.

Системна невизначеність полягає у невизначеності функціоналу, який визначає міру релевантності документу та відповідного пошукового запиту. Розв'язок даного типу невизначеності полягає у комбінуванні ймовірнісного підходу і моделюванням мови.

Розв'язок ситуаційної невизначеності полягає у застосуванні принципу гарантованого результату індивідуально до кожного користувача. Застосування регуляризації дозволяє адаптувати модель до багатьох критеріїв.

1.3 Порівняльний аналіз систем текстового пошуку

Головними характеристиками системи інформаційними слугують:

- модель зберігання даних;
- модель індексування даних;
- алгоритми інформаційного пошуку;
- адаптовані алгоритми загального призначення.

Ключовим компонентом будь-якої системи текстового пошуку є тексові індекси. У таблиці 1.1 відображено зведений список індексних даних (тієї інформації, яку необхідно підхраховувати на етапі всатвки документів у базу даних.

Таблиця 1.1 – Статистичні індекси

№	Характеристика індексу	Тип індексу	Lucene	Sphinx
1	Індекс термів (список входжень)	Термовий	+	+
2	Кількість документів, у яких зустрічається терм	Термовий	+	+
3	Сумарна частота зустрічей терму	Термовий	+	+
2	Кількість термів без повторень	По полям	+	+
3	Сумарна кількість термів з повтореннями	По полям	+	+
4	Кількість термів	По полям	+	+
5	Кількість непорожніх документів	По полям	+	+
7	Кількість документів в індексі, включаючи видаленні	Сегментний	+	+
8	Кількість документів в індексі, відкидаючи видаленні	Сегментний	+	+
9	Кількість документів в індексі, включаючи тільки видаленні	Сегментний	+	+
10	Кількість проіндексованих полів	Сегментний	+	+
11	Кількість термів у полі документу	Інвертація поля	+	+
12	Кількість унікальних термів у полі документу	Інвертація поля	+	+
13	Частота найбільш частого терму у полі терму	Інвертація поля	+	+

Дослідження в області ймовірносних методів визначення релевантності привели до появи функцій оцінювання BM25, DFR. У таблиці 1.2 відображено підтримки ймовірнісних методів у сучасних пошукових системах.

Таблиця 1.2 – Оцінювальні функції на основі ймовірнісного підходу

№	Функція оцінювання	Solr	Sphinx
1	BM25	+	+
1	BM25F	-	-
2	DFR (Дівергенція випадкових моделей)	+	+

Види кодування елементів інвертованого індексу визначають швидкість пошуку. У таблиці 1.3 зображено швидкості декодування індексу.

Таблиця 1.3 – Швидкість декодування індексу (у мільйонах цілих чисел за секунду)

Кодування	Характер реалізації	Вікіпедя	SI	STRIP
LZMA	послідовний	65	69	66
varint-SU	послідовний	296	653	350
varint-SU	SIMD	540	692	568
varint-GB	таблиця масок	847	846	844
varint-GB	SIMD	1283	1333	1272
varint-G8IU	SIMD	1272	1608	1415
varint-G8CU	SIMD	1097	1544	1286

Окрім швидкості декодування, при виборі алгоритмів стиснення індексу, також має значення коефіцієнт систеснення α. Даний коефіцієнт визначає розмір стиснутого індексу відносно не початкового розміру. Зобразимо коефіцієнт стиснення у таблиці 1.4.

Таблиця 1.4 – Кофіцієнт стиснення для різних кодувань інвертовонаго індексу

Кодування	Вікіпедя	SI	STRIP
varint-SU	0.43	0.28	0.28
varint-GB	0.43	0.33	0.33
varint-G8IU	0.43	0.31	0.41
varint-G8CU	0.43	0.31	0.39

Із врахування усіх факторів, робимо висновок, що варіант кодування varint-G8IU - найркащий з точки зору компромісу між швидкістю декодування та коефіцієнтом стиснення.

Висновки до розділу

У даному розділі проведено ретельний аналіз індустріальних стандартів в області текствого пошуку. Також порівняно і розглянуто останні дослідження в області стистення інформаційного індексу. Розглянуто типи індексів і величини, які можуть бути обраховані, беручи до увагу структуру інвертованого індексу.

Підсумувавши, можна, сказати, що існуючі реалізації мають настпупні алгоритмічні недоліки:

а) відсутність стиснення індексу;
б) не оптимальне використання процесорних ресурсів;
в) відсутність механізмів комбінування модальностей;
г) відсутність механізму розширення текстового пошукового запиту.

Таким чином існуючі де-факто стандартні рішення можна покращити за допомогою:

а) застосування кодування varint-G8IU для інвертованого індексу;
б) векторної моделі для розширення пошукового запиту;

в) моделі машинного навчання для визначення фінального ранжування.

РОЗДІЛ 2 Алгоритми інформаційного пошуку

2.1 Векторне представлення слів

Векторне представлення слів використовується з метою застосування векторної арифметике над слова. Таким чином, у векторах кодуються.

2.1.1 Модель Skip-Gram

2.1.1.1 Архітектура нейронної мережі

У даній секції буде розглянуто архітектуру нейронної мережі для векторного представлення слів.

Нейронна мережа із одним прихованим для моделі скіпграм розв'язує штучну задачу у тому сенсі, що нейронна мережа не буде використовуватися у тому напрямі, у якому була натренована. Спражньою метою застосування нейронної мережі є знаходження ваг для прихованого шару. Таким чином, ми матимемо векторні представлення слів.

Дана техніка застосовується для знаходження ознак без вчитаеля для тренування автокодувальника, який виконає стиснення у прихованому шарі і розпаковує у вихідному шарі. Після тренування вихідний шар ігнорується і використовується прихований шар. Таким чином мина знаходимо ознаки без розмічених даних.

Конкретизуємо формулювання штучної задачі для векторного представлення слів. Спробуємо натренувати нейронну мережу для наступної задачі: по даному центральному слову із речення, знаходимо близькі слова і будемо випадково обирати близькі. Під близькими розумуються ті, слова, що знаходяться в межах вікна фіксованого розміру n, тобто n слів та n слів після.

У прикладі на рисунку 2.1 зображено для простоти розмір вікна рівний двом. Синім колором позначено центральне слово. Мережа навчиться із статистики появи кожної із пар.

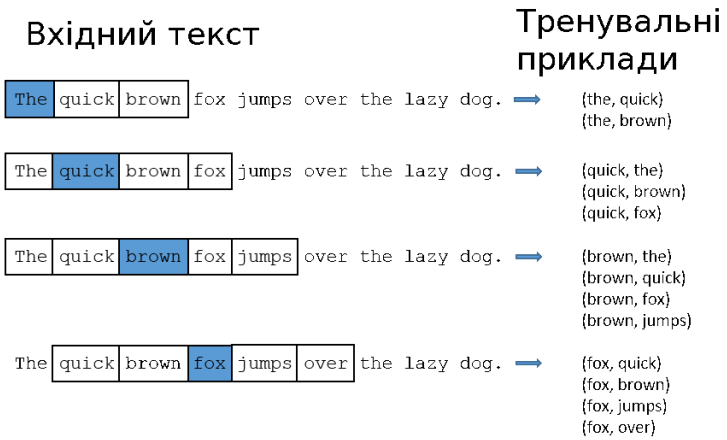

Рисунок 2.1 – Елементи навчальної вибірки

Перейдемо до задачі представлення даних. Для нейронної мережі дані не можуть бути представлені у вигляді тексту. Для цього спочатку знайдемо спосіб представлення слів для нейроннної мережі. Нехай представлено вектор із $|V|$ унікальних слів.

Кожне слово може бути закодоване розрідженим вектором, у якому тільки в одній із позицій знаходиться одиниця. Вихід нейронної мережі - один вектор із $|V|$ компонент, де $|V|$ - розмір словника.

На риснуку 2.2 зображено вигляд структури елементів навчальної вибірки.

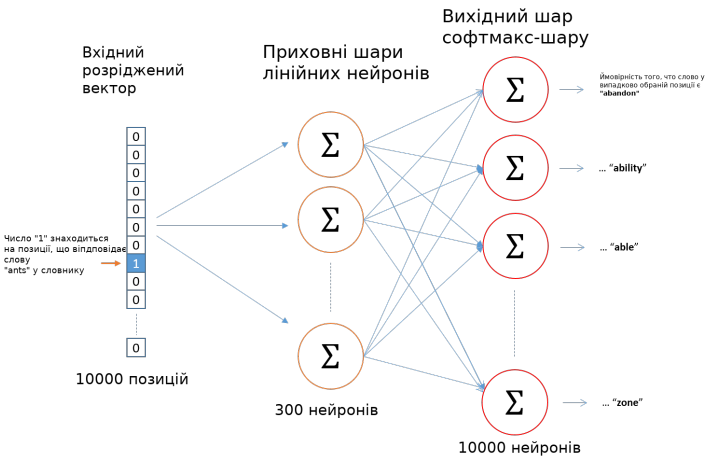

Рисунок 2.2 – Елементи навчальної вибірки

Вихідні ймовірності відносяться наскільки ймовірно знайти слово із словника. Натренуємо нейронну мережу, додаючи словесні пари, що зустрічаються у документах.

Звернімо увагу на відсутність активаційної функції для прихованого шару нейронів. Нейронна мережа тренується на парах слів, вхідний вектор -одинично закодований вектор, що представляє вихідне слово. Коли ми будемо оцінювати нейронну мережу на основі вхідного слова, вихідне слово утворюватиме ймовірнісний розподіл (певна кількість чисел із плаваючою комою).

Опишемо прихований шар. Таким чином навчальний вектор містить 300 ознак. Таким чином, прихований шар представляється за подомогою вагової матриці із $|V|$ рядків та k стовпців. У типовому випадку $k = 300$. Наприклад, Google для моделі на основі новин використовував 300 ознак. Отже, кількість ознак - це гіперпараметр.

Вектор-рядки матриці ваг будут і будуть являти собою векторні представлення слів 2.3.

$$[0 \quad 0 \quad 0 \quad 1 \quad 0] \times \begin{bmatrix} 17 & 24 & 1 \\ 23 & 5 & 7 \\ 4 & 6 & 13 \\ 10 & 12 & 19 \\ 11 & 18 & 25 \end{bmatrix} = [10 \quad 12 \quad 19]$$

Рисунок 2.3 – Множення вхідного слова на матрицю ваг

Для кожного слова вихідний шар має розмір $1 \times k$ розв'яжемо задачу Softmax-регресії, кожен вихідний нейрон (один на слово у словнику). Графічно даний зв'язок зображено на рисунку 2.4.

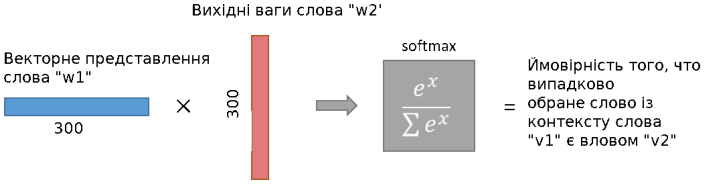

Рисунок 2.4 – Зображення ваг

Кожен вихідний нейрон має вагу, яка множиться на векторне представлення із прихованого шару, а потім підрахоувуємо e^x для результату, як зображено на рисунку 2.4.

Звернімо увагу на те, що нейронна мережа не знає нічого про зміщення вихідного слова відносно вхідного. Вона не навчається ніякому іншому розподілу в залежності від того слово стоїть до чи після центрального слова.

Розглянемо що станеться якщо два різних слова мають схожий контекст. Можна очікувати синоніми “smart” та “intelligent” мають дуже схожі контексти, а також слова “engine” та “transmission” також будуть знаходитися у схожих контекстах. Також описана вище нейронна мережа навчається стемінгу, оскільки різні форми слова можуть вживатися в однакових контекстах. Кількість ваг нейронної мережі дорівнює $|V| * k$, тому необхідно провести кілька прийомів, які могли б мінімізувати вимоги до кількості оперативної пам’яті, оскільки оперативна пам’ять може бути замінена на відео-пам’ять з метою більш повного застосування паралелізму, проте таке прискорення може стати неможливим через надмірні розміри матриць, які необхідно обробляти.

Таким чином, необхідно внести деякі модифікації для описаної вище моделі.Запуск методу градієнтного спуску на великій вережі є повільним, оскільки необхідно обробити ваги мільярди документів, аби отримати точні результати.

Таким чином резонними покращеннями можуть бути:

а) розглядати часті пари слів та фрази як єдине слово;
б) здіснювати відбір вибірки для зменшення кількості тренувальних прикладів;
в) модифікувати цільову функцію за допомогою технік вибору негативних прикладів, що призводить лише до того, что оновлюється лише мала частина ваг.

Вказані вище техніки не тільки прискорюють процес навчання, а і покращують якість отриманої моделі.

2.1.1.2 Модифікація навчальної вибірки

Розглянемо перший пункт із можливих покращень моделі. Якщо розглянути пару “Bostone Globe”, то кожен із термів може мати дуже різні значення в залежності від контексту. Таким чином для кожного слова може бути релевантним більше одного набору контекстів.

У компанії Google натренували нейтронну мережу із 100 мільярдів слів із вибірки Google-новин. Додавання фраз збільшило вибірку на 3 мільйони слів.

Навчання на фразах більш детально розглядається в даній статті [2]. Автори статті розглянули фрази довжини 2, таким чином більш довгі фрази можуть ігноруватися, що означає, для пошукових систем втрату семантики, оскільки іменовані сутності часто можуть складатися із більш, ніж двох слів.

2.1.1.3 Видалення частих слів

Розглянемо другий метод із згаданих можливих покращень моделі. Повернемося до фрази із рисунку 2.1. Таким чином ми можемо ззнайти проблеми із частих слів.

а) пара (“fox”, “the”) нічого не каже про значення слова “fox”, оскільки артиклі зустрічаються майже перед кожним словом;
б) ми матимамо набагато більше прикладів “the”.

Таким чином, видалення частих слів вирішує дану проблему. Для кожного слова із тренувального тексту є шанс, що його буде видалено із тексту. Ймоврність його видалення є пов’язаною із частотою входжень. Нехай дано слово ω_i, $z(\omega_i)$ - частина всіх слів для підрахунку ймовірності із якою слово знаходиться в словнику. Наприклад слово “peanut” можу зустрічатися 1000 разів та у тексті з 1-го мільярду слів. Таким чином, маємо $z(peanut) = 10^{-6}$.

Параметр “sample” контролює як часто відбувається видалення термів. У типовій реалізації він рівний $s = 10^{-3}$. Отже ймовірність того, що слово залишиться у тексті дорівнюватиме:

$$P(\omega_i) = (\sqrt{\frac{z(\omega_i)}{s}} + 1) * \frac{s}{z(\omega_i)} \tag{2.1}$$

Таким чином для $s = 0.001$, маємо:

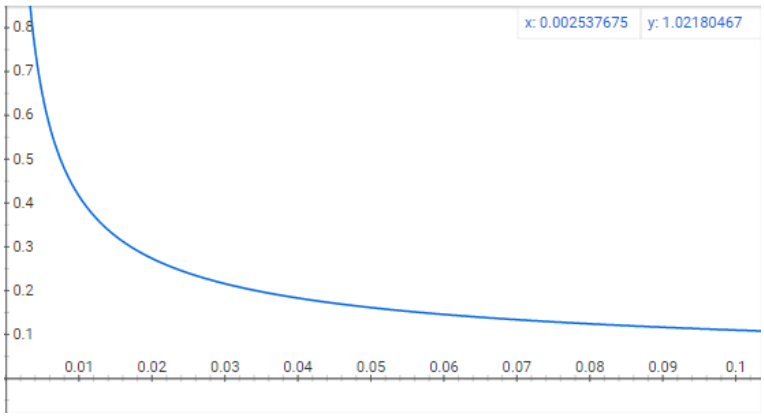

Рисунок 2.5 – Графік залежності ймовірності врахування слова в залежності від частоти

Жодне із слів не повинно домінувати. Тому, на графіку 2.5 нас цікавлять лише малі значення аргменту.

2.1.1.4 Вибір протиріч

Тренування нейронної мережі означає взяття тренувального прикладу і зміну всіх ваг нейронів незначною мірою. Тобто розгляд одного прикляду приводить до зміни всіх ваг.

На практиці у задачах загального призначення використовуються словники із міліардами слів. Для більш конкретизованої області застосування розмір словника можна зменшувати. Коли нейронна мережа тренується на парі слів (“fox”, “quick”), коректний вихід мережі - одинично закодований

вектор.Вихід нейрону, що відповідає слову “quick” рівний 1 та для інших елементів вихід дорівнює 0.

При виборі протиставляючих прикладів, випадково обираємо малу множину із m випадкових слів для оновлення ваг. Це ті слова, для яких бажаним виходом нейронної мережі є 0. Згадаємо, що матриця ваг має розміри $|V| \times k$ Будемо оновлювати ваги позитивного слова “quick”, а також l інших слів, для яких мажаний вихід дорівнює 0. Таким чином потрібно онивати $l * k$ ваг, що при $|V| = 10000, k = 300, l = 5$ становить 0.06% із трьох мільйонів ваг у вигідному шарі мережи.

У прихованому шарі оновлюються лише ваги позитивного прикладу. Негативні приклади обираються із розподілу юніграм.

$$P(\omega_i) = \frac{f(\omega_i)^{\frac{3}{4}}}{\sum_{j=0}^{n} f(\omega_j)^{\frac{3}{4}}} \tag{2.2}$$

2.1.2 Покращення моделі на основі Skip-Gram

Неперервне представлення слів на основі не розміченого корпусу є корисним для задач класифікації, ранжування. Вивчення представлення векторів у діснозначному просторі вивчається із 1954-го року. У більшості випадків дане представлення отримується із не розміченої вибірки, яка являє собою набір документів та ґрунтується на основі частот сумісних частот [3], [4].

Чисельні роботи є пов’язаними із вивченням сементики текстів [5]. Коллоберт та Вестон у 2008 році запропронували використовувати словесні вкладення, використовуючи нейронні мережі шляхом передбачення слова на основі двох слів та двох слів та справа. Після цього було запропоновано модель на основі логарифічної білінійної моделі для ефективного вивчення представлення слів на основі значних об’ємів текстового корпусу [6].

Більшість із згаданих технік представляють кожне слова із словника за допомогою нового вектору без розділення парметрів. У тому числі, ігноруючи внутрішню структуру слів, що є недоліком для морфологічно багатих мов, наприклад, української, турецької, фінської, оскільки більшість слів мають одну або більше форм. При чому деякі форми можуть можуть зустрічатися рідко у порівнянні із іншими формами.

Тому доцільно використовувати n-грами символьного рівня, а слова - за допомогою сум n-грамних векторів. Таким чином, отримана інфомрація про підслова покращаю якість преставлення слів.

Із згаданою областю дослідження, тісно пов'язані дослідження в області симваольних моделей для обробки пови. Такі моделі ігонрують сегментації речень на слова, а покладаються безпосередньо на символи. Такі символи інтенсивно використовуються для визначення частин мови. Такі моделі можуть бути побудовані на основі нейронних мереж із зворотнім розповсюдженням похибки та згорткових нейронних мереж.

Нехай дано словник W розміру $|W|$. Необхідно знайти представлення кожного слова $\omega \in W$.

$$\sum_{t=1}^{T} \sum_{c \in C_t} \log_2 p(\omega_c|t), \tag{2.3}$$

де C_t - контекстні слова для слова ω_t. Таким чином ми визначаємо ймовірність появи контекстного слова ω_c при слові ω_t.

При застосуванні 'softmax' маємо дану ймовірність, задану наступним чином:

$$p(\omega_c|\omega_t) = \frac{e^{s(\omega_t,\omega_c)}}{\sum_{j=1}^{|W|} e^{s(\omega_t,\omega_j)}} \tag{2.4}$$

Зображена вище модель не є адаптованою для описаного вище випадку, оскільки припускає, що ми прогнозуємо лише одне контекстне слова ω_c.

Проблема прогнозування контекстних слів може роглядатися як множина незалежних задач бінарної класифікації. Головною метою є незалежно прогнозувати наявність чи відсутність контекстних слів.

Для кожного слова у позиції t розглянемо всі контекстні слова як позитивні випадки, а негативні будемо випадково обиратимемо із словника. Для кожної позиції контексту c, застосуємо логістичну функцію втрат, отримаємо від'ємний логарифм правдоподібності:

$$\log\left(1 + e^{-s(\omega_t, \omega_c)}\right) + \sum_{n \in N_{t,c}} \log\left(1 + e^{s(\omega_t, n)}\right) \tag{2.5}$$

,

де $N_{t,c}$ - множина негативних прикладів, отриманих із словника. Позначивши логістичну функцію втрат l: $x \to \log\left(1 + e^{-x}\right)$, перепишемо:

$$\sum_{t}^{T} \left[\sum_{c \in C_t} l(s(\omega_t, \omega_c)) + \sum_{n \in N_{t,c}} l(-s(\omega_t, n)) \right] \tag{2.6}$$

Параметризація скорингової функції s між словом ω_t та контекстним словом ω_c - словникові вектори. Позначимо кожне слово ω із словника за допомогою двох векторів u_ω та $v_\omega \in \mathbb{R}^d$. Назвемо їх вхідним та вихідним вектором відповідно. Скорингова функція може бути визначена у вигляді скалярного добутку між словом контекстними векторами: $s(\omega_t, \omega_c) = u_{\omega_t}^T v_{\omega_c}$

Використовуючи різні векторні представлення для кожного, слова, модель на основі скіпграм ігнорує структурні особливості слів, таким чином необхідно вдосканалити скорингову функцію, аби брати до уваги інформацію цього роду.

Представимо кожне слово ω як мішок символьних n − . Додамо спеціальні обмежувальні символи $<$ та $>$ для зазначення початку та кінця представлення відповідно. Наприклад, для $n = 3$ та слова "where" "<wh, whe, her, ere, re>", оскільки були введені 2 додаткових спецсимвола, а також маємо спеціальну послідовність "<where>".

Зазначимо, що послідовність “<her>” відповідає слову “her” відрізняється від триграми “her” із слова “where”. На практиці будемо виділяти все “n-грами” довжини від 3-х до 7-ми. При цьому підході можна брати до уваги різні підмножини n-грам, наприклад всі префікси та суфікси.

Нехай дано словник із n-грам розміру $|G|$. Нехай дано слово ω, зазначимо: $g_\omega \subset 1, .., |G|$ - множина n-грам слова ω. Асоціюємо векторне представлення z_g для кожної із n-грам g. Представимо слово за допомогою суми векторних представлень n-грам:

$$s(\omega, c) = \sum_{g \in g_\omega} z_g^T v_c \tag{2.7}$$

Дана модель дозволяє розділити представлення між словами. Така функція дозволяє отримати представлення навіть для рідких слів.

Для того, аби виконувати умоги по пам’яті, використаємо хеш-функцію, що відображає n-грами у цілі числа від 1-го до K із застосуванням хеш-функції Fowler-Noll-Vo, $FNV - 1a$, $K = 2.10^6$. В ідеалі, слово має бути представлено за допомогою індексу у словнику та множиною захешованих n-грам. Задача оптимізації можна розв’язувати методом стохастичних градієнтів. Нехай дана тренувальная вибірка, що містить T слів і кількість шляхів через дані дорівнює P, розмір кроку в момент часу t дорівнює $\gamma_0(1 - \frac{t}{TP})$, де γ_0 - фіксований парметр.

2.2 Пошук у багатовимірних розріджених просторах

2.2.1 Локалізоване хешування даних у Евклідовому просторі

У даному розділі буде побудовано алгоритм, який із асиптотикою dn^ρ знаходить у d-вимірному простор n найближчих сусідів, де $\rho(c) = 1/c^2 + o(1)$

При $c = 0.25$, $\rho(x) = 0.25$ Більш точно асиптотику алгоритму:

$$t^{c_0 t} n^{1/c^2 + c_1 \log(t)/\sqrt{t}}, \tag{2.8}$$

де t - параметр, що вибирається мінімізації. Множник $t^{c_0 t}$ через те, що ми виконуємо операції над точками у $t-$, де якість збільшується із значенням t. Великі значення t роблять неможливим практичне застосування алгоритму.

Алгоритм покладається на існування локально чутливих функцій. Нехай H - сімейство хеш-функцій, що відображають R^d у дискретний простір U. Для довільних двох точок p та q, розглянемо процес вибору функції $h \in H$ випадково та рівномірно і знайдемо ймовірність того, що $h(p) = p(q)$. Сімейство H називається локально R, CR, P_1, P_2-чутливим якщо для будь-яких двох точок $p, q \in R^d$, при $P_1 > P_2$:

$$||p - 1|| \leq R \implies Pr_H[h(q) = h(p)] \geq P_1 \tag{2.9}$$

$$||p - 1|| \geq cR \implies Pr_H[h(q) = h(p)] \leq P_1 \tag{2.10}$$

Для того, щоб проіслютрувати описану концепції, розглянемо приклад. Нехай маємо бінірні дані(кожна із координат приймає значення 0 та 1). До того ж, нхай відстань між точками p та q обчислюються за формулою відстані Хемінга. Ми можемо скористатися сіместом функцій H, що містить всі проекції вхідних даних на координатні осі. Таким чином H містить всі функції h_i із $\{0,1\}^d$ у $\{0,1\}$, таким чином, що $h_i(p) = p_i$ для $i = 1, ..., d$. Обиранючи хеш-функцію рівномріно із H, означає, що ми обираємо випадкову координату із i, Маємо ймовірність $Pr_h[h(p) = h(q)]$, що є рівною частці координат, для яких p та q узгоджені між собою. Таким чином, $P_1 = 1 - \frac{R}{d}$, при цьому $P_2 = 1 - \frac{cR}{d}$. При $c > 1$, маємо $P_1 > P_2$.

Для ефективної реалізації алгоритму необхідно обрати сімейство H. Різниця між ймовірностьями P_1 та P_2 може бути малою, H не може бути довільним.

Нехай дано сімейство хеш-функцій H із параметрами (R, CR, P_1, P_2), різницю між P_1 та P_2 можна посилити шляхом комбінування кількох функцій.

Введемо додаткові параматри k та L, де L число функцій $g_j(q) = (h_{1,j}(q), .., h_{k,j}(q))$, де $h_{y,j}(1 \leq t \leq k, 1 \leq j \leq L)$, де функції g_j обрано незалежно та рівномірно із H. Вони і є тими хеш-функціями, що будуть застосовуватися для хешування точок.

Структура даних конструюється шляхом заміни кожної точки p із вхідних даних у комірку $g_j(p)$ для $j = 1, ..., L$. Оскільки загальне число комірок може бути великим, розглядатимемо тільки заповнені комірки шляхом хешування значень $g_j(p)$. Таким чином, буде використано $O(nL)$ комірок пам'яті. Комірки містять вказівники на реальні дані.

Процедура вибору хеш-функції:

1. Від $u = 1$ до U, вибремо випадкове зміщення $s_u \in [0, 4\omega]^t$, що визначає сітку $G_u^t = G^t + s_u$ t-вимірному Евклідовому просторі.
2. Оберемо матрицю $A \in M_{t,d}$, де A_{ij} нормально розмоділені величини $N(0, 1)$, помножені масштубючий коефіцієнт $\frac{1}{\sqrt{t}}$. Матриця A означає випадкову проекцію із R^d на R^t.

Процедура обчислення $h()$ у точці $p \in R^d$:

1. Нехай $\hat{p} = Ap$ - проекція точки p на t-вимірний підпростір, що представлено у матриці A.
2. Для кожного значення $u = 1, 2, ...U$.
3. Перевіримо чи $B(\hat{p}, \omega) \cap G_u^t \neq \emptyset$, тобто чи існує $(x_1, x_2, ..., x_t)$, такі, що $p \in B((x_1, x_2, ..., x_t), \omega)$.
4. Якщо ми знайдемо $(x_1, x_2, ..., x_t)$, $h(p) = (u, x_1, x_2, ..., x_t)$, завершимо алгоритм.
5. Повернемо нульовий вектор 0^{t+1}.

Для того, щоб зменшити U, спроектуємо R^d у простір меншої розмірності R^t випадковим чином. Параметр $t = o(\log n, e^t \square o(n)$

Для того, щоб оробити запит q, виконаємо пошук серед комірок $g_1 q, g_2 q, ..., g_L(q)$ і оберемо точки, що там зберігаються. Після копіювання точок, розрохуємо їх відстані до запиту. Можливі 2 стратегії пошуку:

а) перервати пошук після знайдених $L' points$, включаючи дублікати;

б) продовжити пошук поки всі точки із комірок буть скопійовані.

Дві вище згадані стратегії вирішують 2 різні задачі. Перша задача виконує пошук (c, R) найближчих сусідів, в той час як друга стратегія знаходить R наближчих сусідів.

У роботах [7], [8] показано, що перша стратегія при $L' = 3L$, дає розв'язок для рандомізованого пошуку найближчих сусідів із параметрами R, c та δ для деякої ймовірності похибки $\delta < 1$. Для того, аби отримати гарантії, доцільно присвоїти L >= $c_2 * n^\rho$, де $\rho = \frac{\ln 1/P_1}{\ln 1/P_2}$ і $K < C_C \log_{1/P_2} n$. Ці цього слідує, що алгоритм працюватиме пропроційно до n^ρ, що означає сублінійність при $P_1, > \ P_2$, що слідує із побудови. Наприклад, якщо застосувати хеш-функції до бінарних векторів, то отримаємо $\rho = \frac{1}{c}$ [9].

Другою можливою стратегіє можна розв'язати задачу рандомізованого пошуку найближчих сусідів. Ймовірність похибки δ залежить від вибору параметрів k та L [10].

2.2.2 KD-дерева

K-вимірне дерево - структура даних із розбиттям простору для впорядкування точок у k-вимірному просторі. Таке дерево є незбалансованим деревом пошуку для зберігання точок із простору R^k. Вимагає $O(kn)$ додаткової пам'яті. Кожна вершина дерева містить точку та 2 посилання (на ліву та праву верширу відповідну).

Процедура додавання елементів здійснютється шляхом додавання елементів у двійкове дерево пошуку. Видалення елементів розділено на 2 процедури:

а) видалення листової вершини;

б) видалення із перебалансуванням.

Процедура пошуку найближчого сусіда:

1. Починаючи з кореня, рекурсивно рухаємося вниз, шукаючи місце вставки.

2. Якщо досягнуто листову вершину, запам’ятаємо дану вершину як найкраще значеннння.
3. Алгоритм піднімається в горову по стеку рекурсивних викликів.
4. Перевіряємо чи є точки, ближчі точки за дану знайдену найближчку точку.
5. Якщо можуть бути точки, що знаходяться по інший бік розділяючої площини [11].

2.2.3 R-дерево Гільберта

R-дерево Гільберта- це варіант індексування багатомірних об’єктів, таких як прямі, двомірні області, тривимірні об’єкти або об’єкти більших розмірностей. Їх можно трактувати як розширення B + -дерев на багатовимірні об’єкти.

Швидкодія R-дерев залежить від якості алгоритму, що групує дані в прямокутниках. R-дерева використовують заповнюючі простір криві, точніше, криві Гільберта для лінейного упорядкування об’єктів у прямокутниках.

Існує два типи R-дерев Гільберта, одна для статичних даних, інша - для динамічних. У обох випадках для отримання кращого упорядкування багатомірних об’єктів використовуються заповнення простору кривими Гільберта. Это упорядкування ”хороше” в тому сенсі, що воно повинно було б групувати ”схожі” дані в прямокутниках, мінімізуючи площа і периметр цих мінімальних обмежувальних прямокутників (Мінімальний обмежуючий прямокутник). Упаковані R-дерева Гільберта пригодні для статичних даних, оновлюваних дуже рідко або не обновлюваних взагалі.

Динамічні R-дерева Гільберта годяться для змінних даних, де вставки, видалення або поновлення відбуваються в режимі реального часу. Більш того, динамічні R-дерева Гільберта використовують гнучкий механізм розщеплення, що поліпшує обробку приміщень. Кожен вузол має чітко визначену множину множину батьківських вузлів.

Таблиця 2.1 – Кількість запитів за секунду при розмірності, рівній 256

Стурктура даних	Кількість запитів за секунду
KD-Tree	5
Hilbert R-Tree	10
LSH	103

Регулюючи політику розщеплення, за допомогою R-дерев Гільберта можна отримати ступінь обробки простору на бажаному рівні точності. R-дерева Гільберта сортують прямокутники відповідно до Гільбертовых відстаней центрів прямоугольників. (Гільбертова відстань точки рівне довжині кривої Гільберта від початку кривої до точки.). В протилежність цьому інші варіанти R-дерев не мають механізмів контролю над обробкою простору. [12]

2.2.4 Вибір алгоритму пошуку у багатовимірному просторі

Вимоги до вибору алгоритму диктуються наступними факторами:

а) швидкість роботи при збільшенні розмірності до 300 елементів;
б) швидкість роботи при дисковій реалізації;
в) якість пошуку.

KD-дерева годяться тільки для пошуку у просторах малої розмірності. Результати із таблиці 2.1 підтверджують це теоретичне твердження на практиці. Видно, що найбільш доцільним алгоритмом для практичного застосування є LSH. Варто зазначити, що дані у даній таблиці були округлені в менший бік. Без врахування дисперсії вимірювань, на основі даних із таблиці 2.1 стає зрозуміло, що деревовидні структури даних не підходять для практичного застосування.

2.3 Узагальнена реалізація алгоритмів загального призначення у задачі інформаціного пошуку

2.3.1 Узагальнені алгоритми для ефективного знаходження частотного розподілу елементів

2.3.1.1 Загальні терміни

У даній главі розглянуто асоціативні структури у контексті побудови узагальнених алгоритмів, представлено альтернативний підхід до знаходження частотного розподілу елементів, запроновано узагальнений алгоритм на основі сортування та бінарного пошуку, проаналізовано його складність. Швидкодію отриманого алгортму порівняно із шкидкодією асоціативних контейнерів, які базуються на використанні хеш-таблиць та червоно-чорних дерев.

Концепції, Алгоритми, Швидкодія, Вимірювання Програмування є математичною дисципліною. Головним завданням програмування є знаходження ефективного обчислювального базису.

Знаходження частноного розподілу різнорідних елементів є підзадачею статистичного аналізу, застосовується у задачах попередньої обробки інформації, інформаційного пошуку.

До конкретних проявів можна віднести:

- пошук розподілу слів у тексті;
- визначення апостеріорних ймовірностей;
- обчислення функції ранжування Okapi BM25 [13];
- обчислення міри TF-IDF.

2.3.1.2 Узагальнений підхід

Узагальненим програмуванням називається підхід до програмування, при застосуванні якого акценується увага на проектуванні алгоритмів і

структур даних, які працювали б у найбільш загальних умовах без втрати ефективності.

Концепція - опис вимог для одного або кількох типів, заданих у вигляді існування операцій, їх властивостей, атрибутів типів, існуванні функцій на типах, визначених на типах, що розглядаються (відношення, бінарний предикат, повний порядок).

Візьмемо регулярний тип, - тип даних, для якого бінарний предикат рівності, який узгодженний із операціями присвоєння та конструювання:

а) T a = b; assert(a == b);
б) a = b; assert(a == b);
в) T a = b; $\iff$ T a; a = b;
г) Наявність деструктора.

Оскільки предикат рівності надає операцію лінійного пошуку, що приводить до квадратичного алгоритму. Без побальшого алгоритмічного аналізу, можна зробити висновок, що такий алгоритм такої складності для даної задачі не має доцільного практичного застосування. Для прискорення алгоритму, необхідно ввести відношення пошуку. [14] У контексті даної задачі відношення порядку може бути введено:

а) введення хеш-функції із застосовуваних типів у цілі невід’ємні числа;
б) введенням відношення строго порядку на множині елементів, для яких повинен праюцювати алгоритм.

При застосуванні асоціативних контейнерів для знаходження частотного розподілу, операцій:

а) вставка;
б) пошук.

У таблицях 2.2 та 2.3 основні операціями із послідовностями та їх властивості.

Таблиця 2.4 – Операції із відсортованим масивом, що не порушують інваріанту відсортованості

Контейнер	Складність операції	
	Вставка, що не порушує порядку	Бінарний пошук
Відсортований масив	$O(n)$	$\log n$
Червоне-чорне дерево	$O(\log n)$	$\log n$

Таблиця 2.2 – Операції над асоціативними контейнеру, що підримують інваріант

Контейнер	Складність операції	
	Вставка	Пошук
Червоне-чорне дерево	$\log n$	$\log n$
Хеш-таблиця із лінійним зондуванням	$C(\alpha)O(1)$	$C(\alpha)O(1)$
Хеш-таблиця із списками	$O(1+\alpha)$	$O(1+\alpha)$
Зозулине хешування	$O(1)$ амотртизоване значення	$O(1)$ у найгіршому випадку

Таблиця 2.3 – Операції, які не покладаються на інваріанти

Контейнер	Складність операаації	
	Вставка в кінець	Лінійний пошук
Масив	$O(1)$	$O(n)$
Хеш таблиця	$O(1)$	$O(1)$

2.3.1.3 Червоно-чорні дерева

Червоно-чорне дерево - це бінарне дерево пошуку, що містить колір, вказівник на ліву дочірню вершину, вказівник на праву дочірню вершину, вказівник на предка та поле із даними. Над типом даних повинен бути визначений оператор порівння, що відношає слабке строге відношення поярдку.

Якщо не існує такого шляху від кореня до будь-якої вершини, такого шляху, що більше, ніж в 2 рази довший за будь-який інший.

Властивості червоного-чорного дерева:

а) кожна вершина має червоний або чорний колір;
б) корінь завжди має чорний колір;
в) кожна термінальна вершина має чорний колір;
г) якщо вершина має червоний колір, то обидві дочірні вершини мають чорний колір;
д) для кожної вершини все прості шляхи від цієї вершини до містять однакову кількість чорних вершин [15].

Вказані властивості дозволяють побудувати структуру даних, що дозволяє виконувати операції вставки, пошуку, видалення за логарифмічний час [16].

Якщо розглянути найбільш нативну структуру даних(близьку до комп'ютерної архітектури), - масив, то легко бачити, що операції пошуку та вставки протирічать між собою, оскільки вставка у відсортований масив із подальшим зберіганням інваріанту є лінійною операцією. В свою чергу для асоціативних контейнерів висока асимптотична швидкість пояснюється компромісами, на які довелося іти, аби забезпечити одночасну швидкодію обох операцій [17].

Із таблиці 2.5 робимо висновок, що швидкодія алгоритма може залежати більше від місцезнаходження об'єкта в ієрархії пам'яті, ніж асиптотичною складністю. $2^{63} = 9223372036854775808$, проте $\log_2 2^{63} = 63$.

Таблиця 2.5 – Залежність швидкості доступу до пам’яті в залежності від ієрархії

Рівень ієрархії	Швидкість доступу, нс.
доступ до регістру	0.1
доступ до кешу 1-го рівня	0.5
доступ до кешу 2-го рівня	7
доступ до пам’яті	100

Оскільки доступ до кешу 1-го рівня в 200 разів перевищує швидкість доступу до памя’ті, це дозволяє розглядати алгоритми, що потребують асимптотично $n \log n$ операцій, можуть працювати швидше за лінійні.

2.3.1.4 Алгоритм пошуку частостей лінійних у впорядкованій послідовності

Концепції, які виражають сімейства типів, на яких працюють алгоритми над послідовностями.

$$Iterator(T) \triangleq Regular(T) \wedge DistanceType : Iterator \to Integer \wedge \\ \wedge\, successor : T \to T \wedge \\ \wedge\, successorisnotnecessarilyregular \tag{2.11}$$

$$ForwardIterator(T) \triangleq Iterator(T) \wedge regular\_unary\_function(successor) \tag{2.12}$$

$$TotallyOrdered(T) \triangleq Regular(T) \wedge <: T \times T \to bool \wedge \\ \wedge\, total\_ordering(<) \tag{2.13}$$

$$RandomAccessIterator(T) \triangleq IndexedIterator(T) \wedge \\ \wedge BidirectionalIterator(T) \wedge \\ \wedge TotallyOrdered(T) \wedge \\ \wedge (\forall i, j \in T)\; i < j \Leftrightarrow i \prec j \wedge \\ \wedge DifferenceType : RandomAcessIterator \to Integer \wedge \\ \wedge + : T \times DifferenceType(T) \to T \wedge \\ \wedge - : T \times DifferenceType(T) \to T,\; takes\ constant\ time \wedge \\ \wedge - : T \times T \to DifferenceType(T) \wedge \\ \wedge\, takes\ constant\ time \tag{2.14}$$

Розглянемо алгоритм на основі бінарного пошуку. Передумови роботи алгоритму:

а) відкритий діапазон $[f, l)$ є відсортованим за предикатом, який є відношенням повного порядку;

б) f, l - ітератори із довільним доступом;

в) out - вихідний ітератор;

г) тип значення, що зберігається в ітераторі out - пара із значенням, що зберігається у вхідному діапазоні та цілочисельний тип;

д) над ітераторам визначена функція на типах $ValueType$, что визначає тип даних, на які вказує ітератор.

Опишемо алгоритм покроково:

1. Якщо 2 вхідних f та l ітератора співпадають, повернемо вихідний ітератор out.
2. Запустити пошуку верхньої границі($upper\_bound$) на вхідних ітераторах $[f+1, l)$ та значенні, що зберіється у середині f, збережемо знайдену позицію pos.
3. Порахуємо відстань від f до pos.
4. Записати у вихідний ітератор відстань та значення.
5. Збільшити ітератор out на 1 одиницю.
6. Записати границю pos у ітератор f.
7. Перейти на крок 1.

Роглянемо складність алгоритму listing/frequencies_bs_0:

$$T_1(n, l, d) = \sum_{i=1}^{l} T_{ub}(n - \sum_{j=1}^{i} d_j), \tag{2.15}$$

де T_1 кількість операцій порівняння елементів, l - кількість унікальних елементів, n - кількість елементів, що зберігаються, d_j - кількість повторень j-го унікального елементу, $T_{ub}(x)$ - швидкодія алгоритму $upper\ bound$, $T_1(n, l, d) \leq lT_{ub}(n) = O(l \log n)$ Розглянемо лінійний алгоритм:

Передумови алгоритму: [11]

а) відкритий діапазон $[f, l)$ є відсортованим за предикатом, який є відношенням повного порядку;

б) f, l - ітератори із послідовним доступом;

в) out - вихідний ітератор;

г) тип значення, що зберігається в ітераторі out - пара із значенням, що зберігається у вхідному діапазоні та цілочисельний тип;

д) над ітераторам визначена функція на типах $ValueType$, что визначає тип даних, на які вказує ітератор.

Опишемо алгоритм покроково:

1. Якщо 2 вхідних f та l ітератора співпадають, повернемо вихідний ітератор out.
2. Запустити пошуку верхньої границі на основі лінійного пошуку на вхідних ітераторах $[f + 1, l)$ та значенні, що зберіється у середині f, збережемо знайдену позицію pos.
3. Порахуємо відстань від f до pos.
4. Записати у вихідний ітератор відстань та значення.
5. Збільшити ітератор out на 1 одиницю.
6. Записати границю pos у ітератор f.
7. Перейти на крок 1.

Альтернативно можно використати лінійний пошук замість бінарного, у цьому випадку складність алгоритму виражається:

$$T_2(n, l) = n - l, \tag{2.16}$$

T_2 - кількість операцій порівння елементів, n- кількість елементів, що зберігаються, l - кількість унікальних елементів

2.3.1.5 Вимірювання швидкодії та порівняльний а□наліз

2.3.1.6 Фіксований розмір діапазону для рівномірно розподілених значень

Зафіксуємо розмір масиву $1,000,000$ елементів. У якості елементів будемо брати цілі числа. Згенеруємо цілі числа у діапазоні $[0.l_i)$ та зобразимо залежність швидкодії алгоритму разом із попередьої обробкую у варіантах з бінарним та лінійним пошуком, а також порівняємо із підходом а основі асоціативних контейнерів. Асоціативні контейнери:

а) std::unordered_map<int, size_t> - словник на основі хеш-таблиці із однозв’язними списками;

б) std::map<int, size_t> - словник на основі червоно-чорного дерева.

Із рисунку 2.6 зрозуміло урахування часу на сортування алгоритми $frequencies$ та $frequencies\_bs$, працюють до 10-ти разів швидше за червоно-чорне дерево при застосуванні над 32-бітним цілочисельним типом.

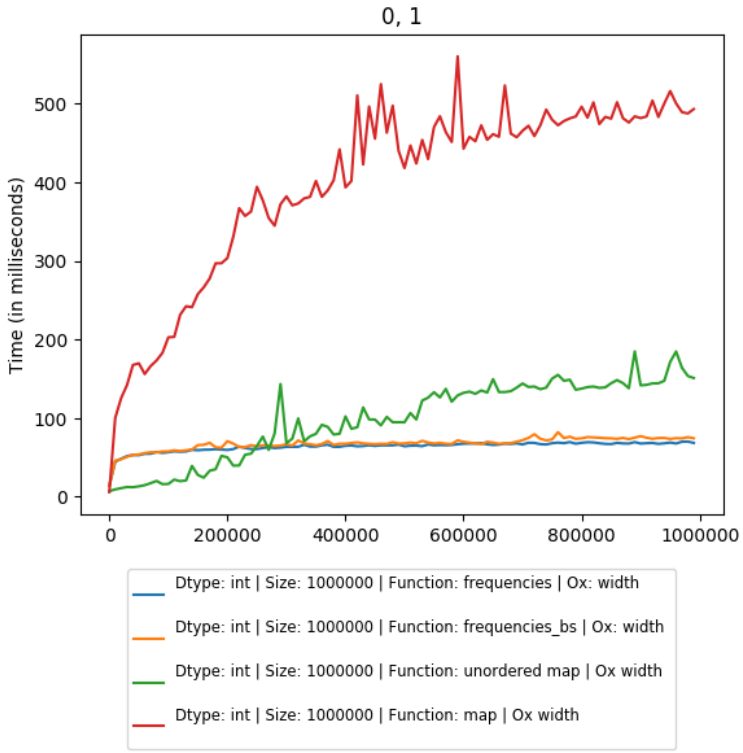

Рисунок 2.6 – Час, затрачений на підрахунок частотного розподілу елементів в залежності від ширину відрізку, на якому генерувались числа. Кількість вхідних елементів(цілих чисел), рівна $1,000,000$

Із рисунку 2.7 зрозуміло урахування часу на сортування алгоритми $frequencies$ та $frequencies\_bs$, працюють до 2.5-ти разів швидше за червоно-чорне дерево при застосуванні над рядком.

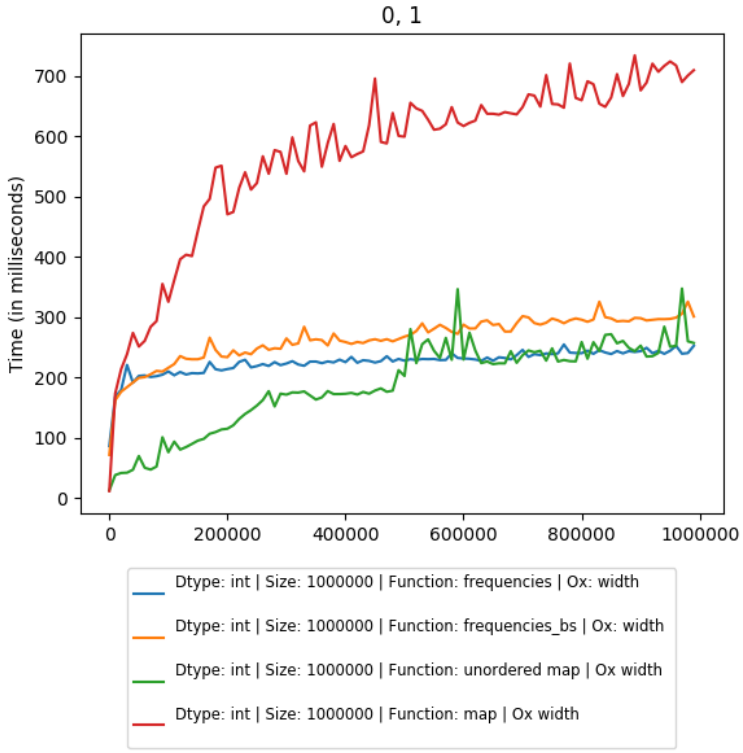

Рисунок 2.7 – Час, затрачений на підрахунок частотного розподілу елементів в залежності від ширини відрізку, на якому генерувались числа. Кількість вхідних елементів(рядків), рівна $1,000,000$,

2.4 Не баєсівська адитивна регуляризація у моделюванні тем великих колекцій даних

Ймовірнісне моделювання тем на основі текстів є потужним інструментом у статистичному аналізі текстів, що ґрунтується на графічних моделях та баєсівському навчанні. Адитивна регуляризація для моделювання тем - новий напівймовірнісний підхід, що дає можливість простіше робити висновки для багатьох випадках із Баєсівським підходом до теорії ймовірностей. Далі буде розглянуто побудову компонент для моделювання тем та продемонстровано можливості не баєсівської регуляризації, спираючись на кілька критеріїв, щоб знайти розріджені різноматінті теми.

Моделювання тем слугує пошуку приховананої тематичної структури колекції текстів та стистнутому представленні кожного документів у вигляді списку тем. Практичні застосування моделювання тем включають в себе інформаційний пошук, класифікацію, категоризацію, сегментування та підсумовування текстів. Моделювання тем поширює свій вплив також і на області, де дані не мають текствого формату: відео, зображення, цифрові сигнали, мережі.

Зі статистичної точки зору, ймовірнісне моделювання тем визначенає кожен топік, базуючись на мультиноміальному розподілі слів і описує кожен документ за допомогою мультиноміального розподілу серед тем.

У сучасній літературі згадуються сотні моделей, адаптованих до різних ситуацій. Ти не менш, більшість моделей надто складні для практичного застосування. Це приводить до того, що у реальних системах застосуються тільки найпростіші моделі, наприклад Латентний ймовірнісний аналіз аналіз тематики та Латентний алгоритм Діріхле. Більшість практичних складнощів є спричиненими баєсівським навчанням, що є домінуючим підходом до моделювання тем.

Баєсівське навчання вагомим теоретичним інтсрументом. Моделювання тем є лише одним із багаточисельних його застосувань. Баєсівський вивід є елегантним для спряженого апріорного розподілу. З точки зору лінгвістики спряжений розподіл Діріхле не є обов’язково накращим вибором, при чому конфліктує із припущенням про розріженість даних. Більш точні не спряжені розподіли вимагають математичних розрухунків, які запутують алгоритм та роблять його асимптотично складним. Розробка багатоцілевих комбінованих моделей є надто складним при застосуванні Баєсівського підходу. При чому існуючи реалізації через свою асиптотичну складність не можуть бути застосованими для великих текстових колекцій, які зустрічаються на практиці. До сьогоднішнього моменту не існує вільного програмного забезпечення, яке дозволило б комбінувати моделі для тематичного моделювання.

З точки зору теорії оптимізації, моделювання тем може розглядатися як спеціальний випадок чисельної стохастичної факторизації матриць. Задача отримання факторизованої презентації текстової колекції є некоректно

поставленою, оскільки має нескінченну множину розв’язків. Типовим підходом до регуляризації у цьому випадку є додавання в цільову функцію обмежень у вигляді адитивних термів.

Адитивна ругуляризації для моделювання тем є напівйтовірнісним підходом, що ґрунтується на класичній(не баєсівській регуляризації). Ми будемо застосувати максимізацію зваженої суми логарифмів правдоподібності. Отриманий критерій не повинен містики логарифмів або мати будь-який ймовірнісний сенс. Задача оптимізації розв’язується за допомогою загального алгоритму максимальною правдоподібності (ЕМ-алгоритм), що може бути легко застосованим до будь-якої комбінації критеріїв. Не баєсівська регуляризація надає набагато простіше виведення для моделювання тем, що раніше розглядалися лише із застосуванням баєсівського підходу. Наприклад, латентне моделювання тем (LDA), може розглядатися як гладкий регуляризатор, що мінімізує дивергенцію Кульбака-Лейбера для кожної теми із розподілу із фіксованим мультиноміальним розподілом. Максимізація дивергенції Кульбака-Лейбера.

2.4.1 Місце мультимодальних даних у інформаційних системах

Мульримодальні дані стають все більш важливими у прикладних областях. Великі колекції даних, що містяться у Вебі, дані із сенсорів гетерогенну зв’язну структуру. У типовому випадку, разом із текстами зберігаються зображення, відео- та аудіо-файли, метадані, що містять інформацію про авторів, часові мітки, посилання. У таких випадках документи розглядуються як мультимодальні контейнери, у яких слова є окремими членами модальностей. Усі модальності є корисними для більш точного визначення тем, так і навпаки, теми є корисними для міжмодального пошуку коли необхідно здійснювати рекомендації для користувачів та заповнювати пропуски у даних.

2.4.2 Адитивна регуляризація для моделювання тем

Онлайн-алгоритми довели свою ефективність для великих колекцій документів, включаючи ті, що працюють із потоками даних. Онлайн алгоритми зараз наявні для таких методів, як PLSA та LDA. Алгоритм, який буде описано далі залишається однаковим як для PLSA та LDA моделей.

Нехай D - скінченна множина текстів і W - скінченний словник термів, що містяться у тексті. Кожен терм може означати як єдине слово або ключову фразу. Кожен документи $d \in D$ є послідовністю термів із словника W. Допустимо, що наявність терму у кожному документі має відношення до прихованої теми t із кінцевої множини T. Колекція текстів розглядається як трійка $(w_i, d_i, t_i), i = 1, ..., n$, взяті із дискретного розподілу $p(w, d, t)$ над скінченним ймовірнісним простором $T \times D \times T$. Терми w_i та документи d_i - спостережувані змінні, в той час як теми t_i - приховані змінні.

Моедлювання тем за ймовірнісним латентним аналізом семантики(англ. PLSA) пояснює ймовірності термів $p(\omega|d)$ в кожному документі $d \in D$, змішуючи ймовірності термів серед тем та ймовірностей тем документів:

$$p(\omega|d) = \sum_{t \in T} p(\omega|t)p(t|d) = \sum_{t \in T} \phi_{\omega t}\theta_{td}, \omega \in W \tag{2.17}$$

Дане представлення безпосередньо слідує із формули повної ймовірності та припущення умовної незалежності $p(\omega|t) = p(\omega|d, t)$, що означає, що кожна тема генерує терми незалежно від теми документу.

Параметри $\theta_{td} = p(t|d)$ та $\phi_{wt} = p(\omega|t)$ утворюють матриці $\Theta = (\theta_{td})_{T \times D}$ та $\Phi = (\phi_{\omega t})_{W \times T}$. Ці матриці є стохастичними. Таким чином, їх вектор-стовпці утворюють дискретний розподіл. Кількість тем $|T|$ зазвичай набагто менша за $|D|$ та $|W|$.

Що підібрати параметри Φ, Θ із колекції документів, необхідно максимізувати логарифм правдоподібності:

$$L(\Phi, \Theta) = \sum_{d \in D} \sum_{\omega \in W} n_{d\omega} \ln p(\omega|d) \to \max_{\Phi, \Theta}, \tag{2.18}$$

де $n_{d\omega}$ - кількість входжень терму $\omega \in W$ у документ d.

Введемо r додаткових критеріїв $R_i O(\Phi, \Theta), i = 1, ..., r$, тобто регуляризатори. Будемо максимізувати регуляризатори окремо від критерію, проте у випадку із регуляризаторами, будемо максимізовувати лінійну комбінацію із коефіцієнтами $\rho_i, i = 1, ..., r$:

$$\sum_{i=1}^{r} \rho_i R_i(\Phi, \Theta) \to \max_{\Phi, \Theta} \tag{2.19}$$

Потім введемо регуляризуючий терм $R(\Phi, \Theta)$ до логарифму правдоподібності $L(\Phi, \Theta)$ і розв'яжемо багатокритеріальну задачу із обмеженнями шляхом скаляризації $r + 1$ критерію:

$$\sum_{d \in W} \sum_{\omega \in W} n_{d\omega} \ln p(\omega|d) + R(\Phi, \Theta) \to \max_{\Phi, \Theta} \tag{2.20}$$

$$\sum_{\omega \in W} \phi_{\omega t} = 1, \phi_{\omega t} \geq 0 \tag{2.21}$$

$$\sum_{t \in T} \theta_{td} = 1, \theta_{td} \geq 0 \tag{2.22}$$

Дана система слідує із умов Каруша-Куна-Такера для локального максимуму рівнянь 2.20 та 2.21. відповідають системі рівнянь із допоміжними змінними, інтерпретовані як умовні ймовірності

$p_{td\omega} = p(t|d, \omega)$:

$$p_{td\omega} = \underset{t \in T}{norm}(\phi_{\omega t}\theta_{td}) \tag{2.23}$$

$$n_{\omega t} = \sum_{d \in D} n_{d\omega} p_{td\omega} \tag{2.24}$$

$$n_{td} = \sum_{\omega \in d} n_{d\omega} p_{td\omega} \tag{2.25}$$

$$\phi_{\omega t} = \underset{\omega \in W}{norm}(n_{\omega t} + \phi_{\omega t}\frac{\partial R}{\partial \phi_{\omega t}}) \tag{2.26}$$

$$\theta_{td} = \underset{t \in T}{norm}(n_{td} + \theta_{td}\frac{\partial R}{\partial \theta_{td}}), \tag{2.27}$$

де оператор норми трансформує дійснозначний вектор $x_t, t \in T$ у вектор $\tilde{x}_t, t \in T$ описують дискретний розподіл:

$$\tilde{x}_t = \underset{t \in T}{norm}(x_t) = \frac{\max\{x_t, 0\}}{\sum_{s \in T} \max\{x_s, 0\}} \tag{2.28}$$

Система 2.23 - 2.27 може бути розв'язана шлляхом застосування чисельних методів. У напростішому випадку - шляхом застосування методу простої ітерації, що еквівалентно EM-алгоритму для тематичних моделей PLSA та LDA.

Шляхом вибору різноманітних регуляризаторів можна побудувати різні моделі. Наприклад, у випадку PLSA відповідає відсутності регуляризації, $R = 0$. У випадку LDA маємо згладжуючий регуляризатор, що мінімізує дивергенції Кульбака-Лейбера $KL(\alpha||\theta_d)$ та $KL(\beta||\phi_t)$.

Вибір рівномірного розподілу у якості розподілів для α та β відповідає випдаку із апріорним розподілом Діріхле у баєсівському підході.

Адитивна регуляризація дає змогу створювати тематичні моделі шляхом вибору вдалої комбінації регуляризаторів.

Наприклад, якщо розділити теми T на 2 підмножини $T = S \cup B$ та налаштувати регуляризатори таким чином, що доменно-спеціальні терми потраплять у множину S, у той час як часті слова потрапляють у множину B. Розрідженість доменних тем $t \in S$ досягнуто за допомогою двох

регуляризаторів, що максимізують дивергенції Кульбака-Лейбера $KL(\alpha||\theta_d)$ та $KL(\beta||\phi_t)$. Гладкість "фонових тем" $t \in B$ досягається мінімізацією обидвох дивергенцій $KL(\alpha||\theta_d)$ та $KL(\beta||\phi_t)$. Врешті-решт коваріація регуляризаторів використовується для того, щоб зменшити кореляцію між стовпцями матриці Ф гарантує різноманітність тем. Таким чином маємо комбінацію регуляризаторів:

$$R(\Phi,\Theta) = -\beta_0 \sum_{t\in S} \sum \omega \in W \beta_w \ln \phi_{\omega t} - \alpha_0 \sum_{d\in D}\sum_{t\in S} \alpha_t \ln \Theta_{td} + \\ +\beta_1 \sum_{t\in B} \sum \omega \in W \beta_w \ln \phi_{\omega t} + \alpha_1 \sum_{d\in D}\sum_{t\in B} \alpha_t \ln \Theta_{td} - , \\ -\gamma \sum_{t\in T} \sum_{s\in T\setminus\{t\}} \sum_{\omega\in W} \phi_{\omega t}\phi_{\omega s}, \tag{2.29}$$

де $\beta_0, \alpha_0, \beta_1, \alpha_1$ - регуляризуючі коефіцієнти.

2.4.2.1 Мультимодальне моделювання тем

Припустимо, що документ може містити не тільки слова, але і терми із інших модальностей. Кожна модальність визначена як скінченна множина термів $W^m, m = 1, ..., M$. Множини W^m попарно не перетинаються.

Прикладами несловесних модальностей можуть слугувати: автори, класи, мітки, часові мітки, посилання на та із інших документів або авторів, іменовані сутності, об'єкти, розпізнані на картинках, користувачі, що прочитали або закачали документи, рекламні банери.

Введемо тематичну модель $p(\omega|d)$ для кожної із модальностей $W^m, m = 1, ..., .M$:

$$p(\omega|d) = \sum_{t\in T} p(\omega|t)p(t|d) = \sum_{t\in T} \phi_{\omega t}\theta_{td}, \theta \in W^m \tag{2.30}$$

Стокастичні матриці $\Phi^m = (\phi_\omega)_{W^m \times T}$ містять ймовірності термів для тем, якщо згрупувати вертикально, отримуємо матрицю Ф розмірності $W \times T$.

Необхідно знайти параметри Φ^m, Ф із мультимодальної колекції. Таким чином необхідно максимізувати логарифм правдоподіббності кожної із m модальнсотей:

$$L_m(\Phi^m, \Theta) = \sum_{d \in D} \sum_{\omega \in W^m} n_{d\omega} \ln_{p(\omega|d)} \to \max_{\Phi^m, \Theta}, \tag{2.31}$$

де $n_{d\omega}$ - кількість входжень терму $\omega in W^n$ у документі d. Розподіли тем у документах Θ є спільними для всіх модальностей.

Складемо зважену суму регуляризаторів $R(\Phi, \Theta)$ до логарифму правдоподібності із отримуємо задачу оптимізації із обмеженнями:

$$\sum_{m=1}^{M} \tau_m L_m(\Phi^m, \Theta) + R(\Phi, \Omega) \to \max_{\Phi, Omega} \tag{2.32}$$

$$\sum_{\omega \in W^m} \phi_{\omega t} = 1, \phi_{\omega t} \geq 0 \tag{2.33}$$

$$\sum_{t \in T} \theta_{td} = 1, \phi_{td} \geq 0, \tag{2.34}$$

Де коефіцієнти регуляризації τ_m означають важливість кожної із модальностей. Локальный максимум задоволняє умови наступної системи

рівнянь:

$$p_{td\omega} = p(t|d,\omega) \tag{2.35}$$

$$p_{td\omega} = \underset{t\in T}{norm}(\phi_{\omega t}\theta_{td}) \tag{2.36}$$

$$n_{\omega t} = \sum_{\in D} \tau_{m(\omega)} n_{d\omega} p_{d\omega} \tag{2.37}$$

$$n_{td} = \sum_{\omega\in d} \tau_{m(\omega)} n_{d\omega} p_{d\omega} \tag{2.38}$$

$$\phi_{\omega t} = \underset{\omega\in W^n}{norm}(n_{\omega t} + \phi_{\omega t}\frac{\partial R}{\partial \phi_{\omega t}}) \tag{2.39}$$

$$\theta_{td} = \underset{t\in T}{norm}(n_{td} + \theta_{td}\frac{\partial R}{\partial \theta_{td}}), \tag{2.40}$$

де $m(\omega)$ - модальність терму $\omega \in W^{n(\omega)}$. Система, записана вище, відповідає умові Куна-Такера.

Для єниної модальності $(M = 1)$ можна застосувати EM-алгоритм із минулого розділу. Таким чином

2.4.2.2 Паралельний EM-алгоритм у режимі онлайн

Розділимо колекцію документів D на фрагменти $D_b, b = 1, ..., B$, і організуємо EM-ітерації, таким чином, що кожен вектор, що представляє документ Θ_d, ітерації продовжуються до тих пір, поки не виконуватиметься збіжність у константній матриці Ф.

Алгоритм 1 не визначає як часто необхідно синхронізувати матрицію Ф на кроках 5-8. Це можна робити після кожного батча а менша часто (наприклад, у тому випадку якщо $\frac{\partial R}{\partial \phi_{\omega t}}$ займає надто багато часу у процесі виконання алгоритму). Гнучкість є важливою у паралельному програмуванні, у тому числі у описуваному алгоритмі, де кілька батчів можуть оброблятися одночасно. У випадку сихнронізацію можна проводити у тому випадку, коли оброблено фіксоване число документів.

Кожен D_b зберігається на диску у окремому файлі і тільки обмежене число батчів може бути завантажено до оперативної пам'яті у фіксований момент часу. Все матриця Θ також цілковито ніколи не зберігається в пам'яті. В результаті, об'єм використаної оперативної пам'яті залишається фіксованим незалежно від розмірів колекції, яку необхідно обробити.

Для того, аби розбити колекцію на батчі та паралельно обробити їх, застосуємо алгоритми $AD-LDA$ та $PLDA+$. Вони вимагають блокування під час оновлення матриці Ф. Даний вид синхронізації сповільнює паралельний алгоритм. Таким чином доцільно змінити сехму алгоритму, аби уникнути зайвих блокувань, що сповільнюють алгоритм. Схема роботи описаного алгоритму відображена на рисунку 2.8.

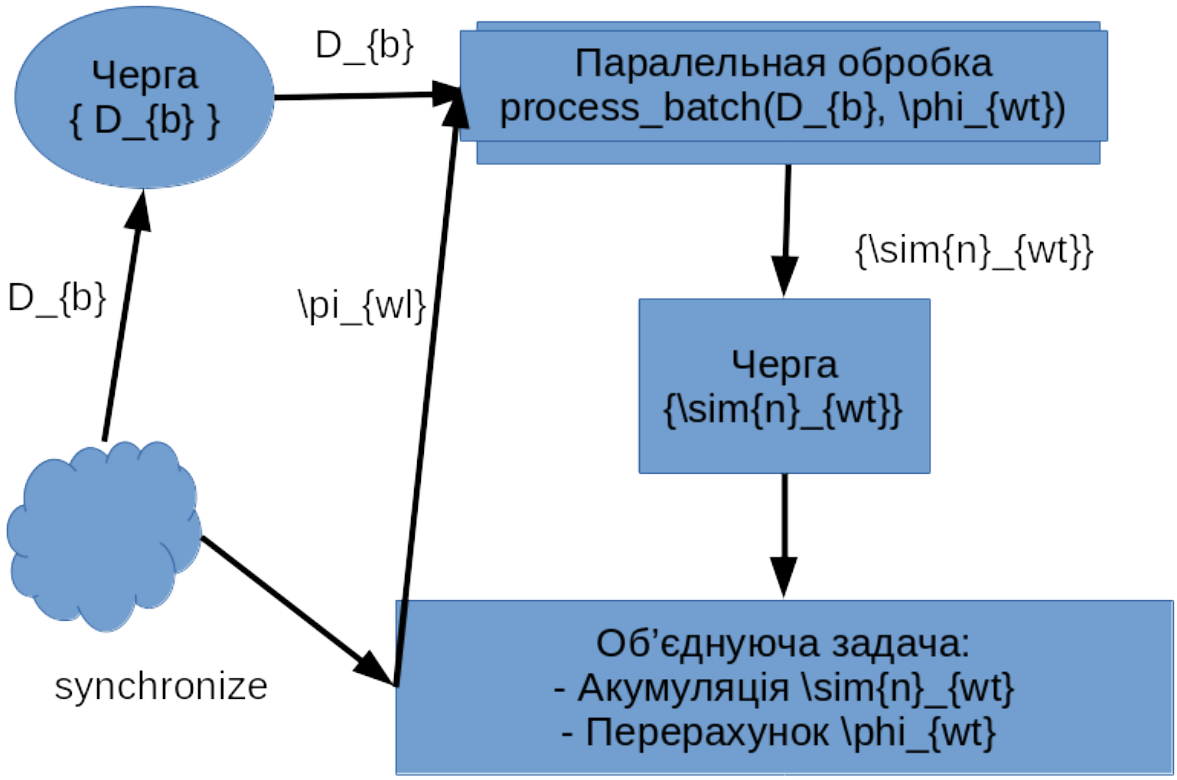

Рисунок 2.8 – Діаграма паралелізму алгоритму обробки батчів

2.4.3 Застосування Баєсівської регуляризації на прикладі реальних даних

У даному експерименті буде побудовано тематичну модель на основі зрізу даних із веб-сайту відповідей на технічні питання. Дана вибірка містить два види модальності:

а) слова;
б) мітки класів.

Можна виділити наступні характеристики вибірки:

а) 25000 документів;
б) навчальна вибірка розбита випадкововим чином перемішана і розита на батчі;
в) 50000 - розмір словника після попередньої оробки;
г) 500 міток.

Мета даного експерименту - побудувати тематичну модель класифікації, яку потім можна використовувати для кластеризації результатів пошуку із метою забезпечення варіативності релевантних документів.

Критеріями якості експерименту є:

а) площа під ROC-кривою(AUC-ROC);
б) площа під кривою PRECISION-RECALL(AUC-PR);
в) частка документів, у яких найбільш ймовірна мітка була виявлена неправильно;
г) частка документів, які не були класифіковано ідеально;
д) середня точність за окрмо взятою міткою.

Варто зазначити, що всі дані характеристики будуть обраховуватися за тестовою вибіркою у якій відсутня інформація про мітки класу.

Параметрами моделі слугують:

а) число тем;
б) число ітерацій по колекції;
в) число ітерацій подкументом;
г) ваги модальностей;
д) коефіцієнти згладжування;

е) коефіцієнти регуляризатора балансування класів.

У таблиці 2.6 відображено результати тематичного моделювання.

Таблиця 2.6 – Кількісні результати моделювання тем

AUC - ROC	Одна помилка	Є помилка	Середня точність	AUC-PR
0.964	64.3%	50.8%	71%	0.211

Висновки до розділу

Векторне представлення слів та фраз довзоляє отримати за попомогою контекстних слів отримати семантичне представлення слів. Таким чином, слова та фрази, які вживаються у схожих контекстах, є близькими у векторному просторі за косинусоїдальної метрикою близькості.

Для швидкого знаходження частотного розподілу небхідно, щоб виконувалася одна із двох умов:

а) була визначеною хеш-функція над елементами, для яких треба знайти частотний розподіл;

б) була визначеною операція порівняння над елементами, а також окрім транзитивності, необхідно аби із еквівалентності слідувала рівність елементів, для яких необхідно знайти частотний розподіл.

Отримана пара алгоритмів є корисною на практиці, оскільки:

- у базах даних дані зберігаються у частково або повністю відсортованаму вигляді, що прискорює алгоритм сортування;
- час, затрачений на попередню обробку даних, компенсується швидкодією алгоритмів на відсортованих послідностей, що компенсує відносну повільність операції сортування.

Підсумуємо, що головними критеріями для вибору типу багатовимірного індексу є:

а) швидкість побудови індекси;

б) швидкість вставки;
в) швидкість пошуку;
г) відсутність вимоги отримання абсолютно точного результату у 100% випадків;
д) швидкодія дискової реалізації.

Із врахуванням вище вказаних вимог та на основі аналізу, проведеного в даному розділі, найбільш доцільним вибором є локалізоване хешування даних (LSH).

Також у цьому отримана пара алгоритмів для пошуку частотного розподілу елементів. Алгоритми працюють не тільки для чисельних типів та рядків, а для типів, над якими задана операція ”менше”, яка у парі із елементами, над якими виконуються алгоритми, відповідає концепції ”TotallyOrdered”(еквівалентність означає рівність).

Крім того алгоритми не обмежуються певним типом контейнерів, а вимагають використовувати ітератори із випадковним та послідовним доступом відповідно.

РОЗДІЛ 3 Система мультимодального пошуку

3.1 Загальна характеристика системи

Система мультимодального пошуку призначена для застосування над документами, що містять дані, які скаладаються із багатьох полів.

Головними компонентами даної системи є:

а) модель нормалізації даних;
б) модель розширення текствого запиту;
в) ймовірнісна модель текстового пошуку;
г) модель машинного навчання.

На рисунку 3.1 зображено діаграму робити розробленої системи та зв'язок між її компонентами. На вхід системи подається пошуковий запит, а вона повертає релевантні документи.

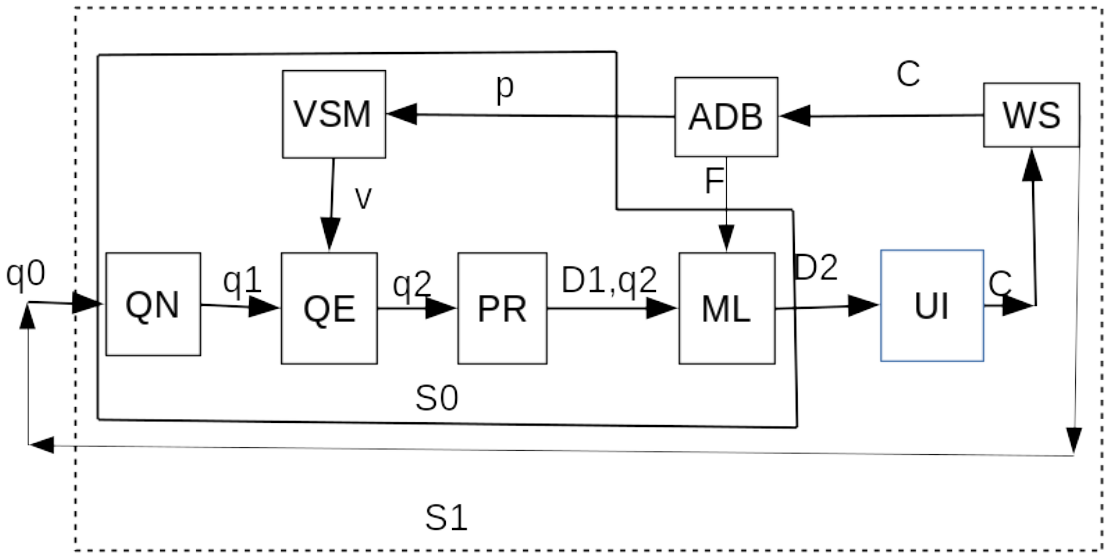

Рисунок 3.1 – Діаграма робити системи мультимодального пошуку

Опишемо умовні позначення із вище згаданої діаграми, де q_0 - пошуковий запит, QN - модуль нормалізації вхідного запиту, q_1 - нормалізований пошуковий запит, QE - модуль розширення вхідного

пошукового запиту, q_2 - розширений нормалізований запит користувача, PR - ймовірнісна модель текстового пошуку, $D1$ - розширений список документів, ML - модель машинного навчання, $D2$ - множина релевантних документів, UI - користувацький інтерфейс, C - інформація про кліки на документи, WS - веб-сервер, ADB - аналітична база даних, що зберігає інформацію про дії користувачів, p - дані для машинного навчання, а саме оновлення векторної моделі представлення слів, VSM - модель векторного представлення слів для розширення слів, v - векторний простір, у якому здійснюється пошук слів, F - матриця ознак. Контур $S0$ показує структуру та внутрішні зв'язки в розробленій моделі мультимодального пошуку, а в контурі $S1$ - зв'язок із іншими компонентами системи.

3.1.1 Опис основних можливостей

Описана в минулому розділі схема роботи системи дозволяє вирішувати широкий діапазон задач:

а) повнотекстовий пошук;
б) пошук по документам;
в) тематичне моделювання документів;
г) виправлення помилок у текстових запитах.

Варто зазначити, що відсутність залежності системи від конкретної структури документу дає можливості різнорідного використання даних.

Ось лише кілька можливих прикладних застосувань:

а) пошук новин;
б) пошук записів всередині блогу;
в) пошук товарів у Інтернет-магазині;
г) тематичне моделювання серед товарів (визначення категорій товарів серед яких потрібно здійснювати пошук);
д) пошук відео

3.1.2 Зворотній зв'язок із користувачами

Запишемо формулу для Okapi BM25 [18]:

$$score(D,Q) = \sum_{i=1}^{n} \omega_i \cdot \frac{f(q_i,D)\cdot(k_1+1)}{f(q_i,D)+k_1\cdot\left(1-b+b\cdot\frac{|D|}{avgdl}\right)} \tag{3.1}$$

$$\omega_i = IDF(q_i) = \log\frac{N-n(q_i)+0.5}{n(q_i)+0.5}, \tag{3.2}$$

Необхідно змінити значення цільового функціоналу. Це можна зробити шляхом модифікації формули, так і шляхом розширення пошукових запитів, імутуючи навчання з учителем.

Таким чином для досягнення поставленої мети, процес формування вибріки буде наступни:

1. Користувач вводить запит.
2. Система надає результати користувачу для перегляду.
3. Користувач ідентифікує релевантні документи.
4. Запит розширюється словами із знайдених релевантних документів, що мають найбільшу вагу.
5. Нові терми додаються до пошукового запиту.

3.1.3 Мова опису записів

За основу мови опису, подібно до Solr, було обрано XML із власною структуруою тегів. Мова розмітки записів ґрунтується на основі XML. Оскільки XML - мова розмітки ієрархічних даних. Це дає можливість побудувати ієрархію із метаданих, що описують структуру збереження та індексування реальних даних. У таблиці 3.1 зображено правила побудови ієрархії даних.

Таблиця 3.1 – Опис мови декларування структури бази даних

Тег	Доступоні параметри	Можливі внутрішні теги
database	name	table, index
table	name, engine	column
column	name, type, size, index	-
index	name, table, column, type, n, compression	-

Кожна мова має набір вбудованих типів, на основі яких будуються користувацькі. Оскільки всі дані у комп’ютерах представлені у числовому вигляді, це означає, що необхідно забезпечити механізми для роботи із числами фіксованої ширини та їхніми послідовностоями. Обрано наступні розміри: 8, 32 та 64 біт, оскільки 8 біт достатньо для кодування символів, 32 біти достатньою для зберігання широкого діапазону практично застосовуваних значень значень. 64-бітні числа застосовуються насамперед у випадку, якщо розмір адресного простору досягає чотирьо та більше гігабайт.

Підтримувані типи стовпців зображено у таблиці 3.2.

Таблиця 3.2 – Опис мови декларування структури бази даних

Тег	Параметр	Можливі значення
database	name	[0-9a-zA-Z_-]+ (унікальне значення)
table	name	[0-9a-zA-Z_-]+
table	engine	flatten
column	name	[0-9a-zA-Z_-]+
column	type	int64_t, int32_t, uint8_t
column	size	[1-9][0-9]+
index	name	[0-9a-zA-Z_-]+
index	table	[0-9a-zA-Z_-]+ (повинна бути створена таблиця із даним ім’ям)
index	column	[0-9a-zA-Z_-]+ (повинно існувати поле у таблиці із заданою назвою)
index	type	inverted_term_int64_t, inverted_term_int128_t, inverted_ngram_int32_t, inverted_ngram_int64_t, rb_tree
index	n	1-64 (у контексті n-грам)
index	compression	none, varint-G8IU

На рисунку 3.2 зображено один із можливих варіантів розмітки таблиці товарів магазину із метою подальшого індексування. При цьому для кожного із полів будуються 2 типи індексів. Один для слів, а другий - для триграм.

```
<database name="store">
  <table name="question">
    <column name="title" type="uint8_t" n="128">
      <index type="inverted_term_int64_t"></index>
      <index type="inverted_ngram_int64_t" n="3"></index>
    </column>

    <column name="description" type="uint8_t"  n="128">
      <index table="question" column="description" type="inverted_term_int64_t"></index>
      <index type="inverted_ngram_int64_t" n="3"></index>
    </column>

    <column name="brand" type="uint8_t" n="128">
      <index table="question" column="brand" type="rb_tree"></index>
    </column>

    <column name="created_at" type="uint64_t"></column>
  </table>
  <index table="question" column="created_at" type="rb_tree"></index>
</database>
```

Рисунок 3.2 – Таблиця питань у базі Інтернет-магазину

3.2 Індексування даних

Будь ласка, допоможіть додаванням доречних внутрішніх посилань або покращенням розмітки статті. (грудень 2012)

Індексування — присвоєння документу набору ключових слів або кодів, які слугують вказівником змісту документа і використовуються для його пошуку. Слід не плутати поняття «індексування» та ”індексацію, оскільки ці поняття різні. Індексування — процес перекладу змісту документів із природної мови на штучну інформаційно-пошукову мову (ІПМ), в результаті чого створюється пошуковий образ документа (ПОД) і пошуковий образ запиту (ПОЗ). У такий спосіб відбувається згортання інформації, що знаходиться в документі, і перетворення її на ІПМ у вигляді індексу, рубрики,коду (класифікаційною мовою) або дескриптора, ключового слова (дескрипторною мовою).

Індексацією називається система і сукупність позначень, прийнята для документної класифікації. Вона виконує кілька основних функцій: закріплює логічну структуру класифікації, виступає засобом зв’язку між діленнями таблиць, рубриками АПП, відділами на книжкових полицях при

систематичній розстановці, засобом запису результатів систематизації в бібліографічних записах, у самих виданнях тощо.

3.3 Зв’язок індексів та метрик

Текстовий пошуковий індекс допускає втрати інформації. Ці врати еквівалентні частковому стемінгу слів. Ці втрати не впливають на якість пошуку, проте до двох разів прискорюють використовувані узагальнені алгоритми. У таблиці 3.3 відображено зв’язок індексів із функціоналом для ймовірнісного пошуку.

Таблиця 3.3 – Зв’язок текстових індексів із функціоналом системи

Тип індексу	Функціонал релевантності
inverted_term_int64_t	Okpai BM25, BM25F для термів, LDA
inverted_term_int128_t	Okpai BM25, BM25F для термів, LDA
inverted_ngram_int32_t	Okpai BM25 BM25F для n-грам
inverted_ngram_int64_t	Okpai BM25 BM25F для n-грам

3.4 Застосування OKAPI-BM25 для мультимодального пошку

Для прикладу візьмемо вибірку із Інтернет-магазину “Homedepot”.

Евристики вибору підмножини документів:

а) вибір останніх n документів;

б) вибір серед k релевантних термів

Спростимо структуру документу до наступних текстових модальностей:

а) назва товару;
б) опис товару;
в) бренд.

З’ясуємо чи можливо до даної вибірки застосувати ймовірнісну модель текстового пошуку.

Рисунок 3.3 – Діаграма робити системи мультимодального пошуку

Введемо матрицю ознак: O_i^0 - вектор коефіцієнтів Okapi BM25 при використанні слів у якості термів, O_i^3 - вектор коефіцієнтів OKAPI BM25 при використання триграм. Нижній індекс відповідає номеру стовпчика. Таким чином маємо наступну матрицю ознак $O_1^0\ O_1^3\ O_2^0\ O_2^3\ O_3^0\ O_3^3\ O_4^0\ O_4^3$. У таблиці 3.4 відображено якість пошук. Чим менше корінь із середньквадартичного відхилення $\sqrt{\sigma}$, тим більш якісно відбувається визначення міри релевантності текстових документів.

Таблиця 3.4 – Коірнь із середньоквадратичне відхилення

Алгоритм	$\sqrt{\sigma}$
Дерево рішень (без розширення набору ключових слів)	0.51
Дерево рішень (з розширенням набору ключових слів)	0.50
Ліс рішень (без розширення набору ключових слів)	0.459
Ліс рішень (з розширенням набору ключових слів)	0.445
CatBoost (без розширення набору ключових слів)	0.456
CatBoost (з розширенням набору ключових слів)	0.435

Отримані результати свідчать про те, що розширення запита на основі векторної моделі із пошукових запитів дає значний приріст у якості пошуку.

Висновки до розділу

У даному розділі описано систему мультимодального пошуку, а також аргументовано покращення існуючих алгоритмів та їх роль у якісному розв’язку проблеми мультимодального пошуку.

Також у даному розділі було розв’язано задачу оцінки релевантності текстових документів, яка у загальному випадку потребує окремого дослідження, що свідчить про високу якість розробленої системи.

В даному розділі:

а) описано мову опису структури даних;
б) описано деталі реалізації мультимодального пошуку;
в) показано місце тематичного моделю у задачі текстового пошуку;
г) оцінено кількісні показники якості пошуку;
д) оцінено вплив пошку.

РОЗДІЛ 4 Керування стартапом проекту

4.1 Інформаційна карта проекту

Назва проекту - “Multimodal Text Search System”. Проект являє собою пошуку систему, яка може представлятися у якості сервісу для кінцевих користувачів.

Необхідні матеріальні ресурси:

а) керований доступ до панелі керування обчислювальними ресурсами;

б) юридична особа.

Інтелектуальні ресурси:

- програмісти ядра системи із знанням математики;
- веб-розробники;
- спеціаліст із маркетингу;
- співробітники служби пітримки;
- відділ лінгвістики;
- дизайнер.
- операційний розробник із підтримки програмного забезпечення.

Необхідні фінансові ресурси: гроші, достатні для оплати ринкової зарплати протягом 3-х років, а також для оплати фунціонування одного веб-сайту, маркетинговий бюджет у розмірі 50 000$.

Проект вирішує проблему, яка характеризується тим, що сучасні бази даних не надають достатніх можливостей для повноцінного текстового пошуку. Існуючі спеціалізовані системи не надають можливості працювати із мультимодальними даними, до того ж, оскільки вони написані із використанням мов програмування, які не мають повноцінного доступу до системних викликів операціної системи та високошвидкісних процесорних операцій та мають вбудоване автоматичне збирання мусору, таким чином, реалізації мають в десятки разів меншу швидкодію, ніж можливо. Таким чином, оплата стороннього сервісу може виявитися дешевшою за плату,

необхідну для функціонування самостійно розгорнутої відкрітої пошукової системи.

Головні цілі проекту:

а) побудова системи мультимодального пошуку на основі ефективних алгоритмів текстового пошуку;
б) максимальна утилізація наявних обчислювальних ресурсів;
в) якісні метрики отриманої системи мають переважити існуючі системи;
г) робастний пошук текстових документів.

Очікувані результати:

а) досягнуто усі головні цілі проекту;
б) наявність передплачених річних та більш довгострокових передплат від користувачів ресурсу;
в) постійний притік нових користувачів;
г) розроблені технології дозволяють вдосконалювати пошукові метрики без деградації швидкодії;
д) система оновлень не наносить шкоди функціонуванню запущених систем.

4.2 Команда проекту

а) засновники проекту;
б) 5 програмістів із знання математики;
в) 3 спеціалісти із пошукових метрик;
г) 2 веб-розробника;
д) маркетолог.

Завданням засновнику проекту є контроль за якістю програмного продукту, підтримуваністю кодової бази, правильним використанням обчислювальних ресурсів хмари, у які зберігаються віртуальні машини.

Завданнями програмістів є постійне вдосконалення швидкодії та підртримуваності існуючої системи, а також вдосконалення реалізованих алгоритмів, а також за необхідності - заміна алгоритмів на більш якісні.

Завданнями спеціалістів із пошукових метрик є вдосконалення функції провдоподібності, яка застосувується при моделюванні тем, інформаційному пошку.

Завданням веб-зробників є розробка користувацьких інтерфейсів для роботи із пошуковою системою, безпечна обробка Інтернет-платежів.

Завданням маркетолога є пошук цільової аудиторії проекту, формулювання портрету ідеального користувача, аналіз вартості та ефективності рекламних кампаній,озробка планів по освоєнню маркетингових бюджетів, огляд створених тарифних планів та доповленостей використання ресурсів.

4.3 Бізнес-модель CANVAS

Модель CANVAS застосовується для опису поточної та майбутньої стратегії, для стратегії розвитку новостворених організацій, для переорієнтації стратегії розвитку діючих організації, розбору існуючої моделі керування з метою знаходження слабких місць/прогалин в діяльності організації та пошуку нових точок для зростання. Авторами, творцями бізнес-моделі CANVAS у 2008 році стали: Олександр Остервальдер – швейцарський бізнес-теоретик та Ів Пін’є – бельгійський вчений і професор інформаційних систем управління. Після чого модель стрімко поширювалась і зараз застосовується викладачами, студентами відомих бізнес-шкіл, університетів: Гарвард, Стенфорд, Колумбія, Берклі.

Далі розглянемо компоненти бізнес-моделі системи мультимодального текствого пошуку.

а) ключові партнери:
   1) інвестори;
   2) рекламні платформи;
б) ключова діяльність:

1) розробка алгоритмів інформаційного пошку;
2) розробка алгоритмів пошуку серед багатовимірних розріджених даних;
3) розробка високоефективних дискових індексів;
4) адапутвання розроброблених алгоритмів для ефективної роботи із диском;
5) розробка веб-платформи для продажу пошукової системи;

в) дистрибуція:
1) підписка;
2) підписка та індивіальна підтримка;
3) довгострокова підписка та розробка розширення функціоналу за вимогами клієнта;

г) ключові ресурси:
- інтелектуальні ресурси (співробітники);
- клієнтська база;

д) ціннісна пропозиція:
- якісний та швидкий пошук при мінімальних витратах ресурсів (трудових та матеріальних);
- маштабованість;

е) відносини з користувачами:
- онлайн-спілкування;
- онлайн-конференції з великими клієнтами;
- особисті тренінги;

ж) стуктура витрат:
- підтримка веб-сайту;
- підтримка делегованих віртальних машин для пошукових систем;
- маркетинговий бюджет;
- витрати на розробку;

з) стуктура доходів:
- підписка;
- корпоративні тренінги.

4.4 Аналіз ринкових можливостей запуску стартап-проекту

У таблицях 4.1 та 4.2 відображено характеристики потенційного ринку та можливих клієнтів.

Таблиця 4.1 – Попередня характеристика потенційного ринку стартап-проекту

№	**Показники стану ринку(найменування)**	**Характеристика**
1	Кількість головних гравців, од.	5
2	Загальний обсяг гравців, $	500 млрд.
3	Динаміка ринку	Попит зростає, проте пропозиція не збільшується
4	Обмеження для входу на ринок	Наявність якісної технології
5	Специфічні вимоги до стандартизації та сертифікації	Відсутні
6	Середня норма рентабельності в галузі	100%

Таблиця 4.2 – Характеристика потенційних клієнтів

№	**Потреба, що формує ринок**	**Цільова аудиторія(цільові сегменти ринку)**	**Відмінності у поведінці різних потенційних цільових груп**	**Вимоги користувачів до товару**
1	Необхідність пошуку релевантних товарів	Інтернет-магазини	Різні магазини мають різні кількості різних типів товарів	Необхідно знаходити різноманітні категорії товарів (забезпечувати варіативність пошукових результатів)
2	Необхідність пошуку релевантних записів	Публічні та приватні блоги, портали новин, інформаційні сайти	Різні ресурси мають різну структуру даних	Пошук повинен враховувати інтереси користувачів а також застарілість чи новизну даних

Фактори можливостей та загроз зображено у таблицях 4.4 та 4.3.

Таблиця 4.3 – Фактори загроз

№	Фактор	Зміст загрози	Можлива реакція компанії
1	Здорожчання послуг хармарних обчислень	Здорожчання цін на DigitalOcean там Amazon приведе до здорожчання тарифів і можливих втрат, пов’язаних із довготривалими підписками	Можлива реакція компанії
№	Втрата ключових клєнтів	Приводить до зменшення доходів	Дізнатися причину втрати

Таблиця 4.4 – Фактори можливостей

№	Фактор	Зміст можливості	Можлива реакція компанії
1	Співробітництво із розвинутими Інтернет-Магазинами	Можливість отримати дохідного клієнта	Підлаштовування платформи під вимоги клієнта
2	Публікація нового стандарту в області пошуку	Можливість швидко адаптувати останнє дослідження	Покращення якості пошку

Факт наявності конкренції нічого не каже про проблеми, які ця конкуренція може породити для компанії, тому доцільніше вводити ступеневі рівні конкуренції на ринку. У таблицях 4.5 та 4.6 відображено ступеневий аналіз конкуренції на ринку.

Таблиця 4.5 – Ступеневий аналіз конкуренції на ринку

№	Особливості конкуретного середовища	В чоему проявляється дана характеристика	Вплив на діяльність підприємства (можливі дії компанії, що бути конкуретноспроможною)
1	Олігополія	Існує 2 відкриті пошукові системи: Apache Solr, Elastic Search	Характеризують мінамльно допустиму якість продукту
2	Міжнародний рівень конкуретної боротьби	Цифрові продукту інформаційного пошуку не мають кородонів	Серйозне ставлення до стійкості системи, рівня безпеки та якості пошуку
3	Галузева конкуренція	Конкуренція проходить у галузі інформаційного пошуку	Бути золотим стандартом у галузі
4	Товарно-видова конкуренція за видами товарів	Наявність функціоналу визначає потенціний розмір бази користувачів	Постійне спостереження за потребами клієнтів
5	За характером конкурентних перваг: цінова/нецінова	Якість пошуку є конкуретною перевагою, водночас товар може обходитись дешевше, ніж альтернативні варіанти	Розширення функціоналу, вдосконалення математичних моделей при сталих цінах
6	За інтенсивністю - марочна конкуренція	Ці не можуть бути тривалий час меншими за вартість хмарних обчислень	Вихід на нових великих користувачів

Таблиця 4.6 – Аналіз конкуренції за М. Портером (Складові аналізу)

Назва характеристики	**Характеристика**
Прямі конкуренти	vespa.ai
Потенційні конкуренти	Goolgle, Yandex, Microsoft, ElasticSearch, Amazon, Solr
Постачальники	Amazon, Digital Ocean, Google
Клієнти	Home Depot, StackOverflow, The New York Times, The Economist
Товари-замінники	Solr, Elastic Search, MySQL, PostgreSQL

У таблиці 4.7 відображено висновки по аналізу конкуренції, а у таблицях 4.8 та відображено обґрунтування факторів конкурентноспроможності та виконано порівняльний аналіз сильних та слабких сторін.

Таблиця 4.7 – Аналіз конкуренції за М. Портером (Висновки)

Назва характеристики	**Характеристика**
Прямі конкуренти	Прямі конкуренти - дрібні компанії
Потенційні конкуренти	Потенційні конкуренти - компанії-гіганти
Постачальники	Постачальники хмарних потужностей - потенційні конкуренти
Клієнти	Для клієнтів якісний пошук - ядро їхньої системи
Товари-замінники	Товари-замінники працюють повільно та демонструють низьку якість пошуку

Таблиця 4.8 – Обґрунтування факторів конкурентноспроможності

№	Фактор конкурентноспроможності	Обґрунтування (чинники, що роблять фактор для порівння конкурентних проектів значущим)
1	Застосування алгоритмів машинного начання для пошуку	Дає можливість розуміти семантику пошукового запиту
2	Лінійна маштобованість	Дає можливість адаптуватися до зростаючих об’ємів даних
3	Низька затримка у 99%-х випадків	Асинхронна модель дозволяє рівномірно розподіляти затримки на по всім з’єднанням
4	Динамічна структура функції релевантності	Дає можливість адаптувати пошукову систему під універсальні потреби

Таблиця 4.9 – Порівняльний аналіз сильних та слабких сторін

#	Фактор конкурентноспроможності	Бали 1-20	Рейтинг відносно vespa.ai						
			-3	-2	-1	0	1	2	3
1	Застосування алгоритмів машинного начання для пошуку	20							+
2	Лінійна маштобованість	20							+
3	Низька затримка у 99%-х випадків	20							+
4	Динамічна структура функції релевантності	20							+

SWOT-аналіз старартап-проекту:

а) сильні сторони:
   1) швидкодія;
   2) простота налаштування розподіленого середовища;
   3) врахування семантики документів;
   4) мультимодальні метрики релевантності;
   5) ефективне стиснення інформації;
   6) малі затримки в обробці запитів;
   7) варіативності релевантних пошукових результатів;

б) слабкі сторони:
   1) відсутність можливості додавання програмованих розширень;
   2) підтримка однієї платформи;
   3) необхідність перевантажувати систему для застосування деяких нових налаштувань;

в) можливості:
   1) можливість лінійно маштабуватися відносно кількості користувачів;
   2) незалежність швидкодії системи від кількості клієнтів;
   3) можливість вибірково встановлювати оновлення пошукової системи;
   4) можливість покращувати систему за рахунок перехресного використання даних;
   5) можливість стати стандартом де-факто в області мультимодального пошуку;
   6) можливість розширити асортимент наданих послуг;

г) загрози:
   1) здорожчання послуг постачальника віртуальних машин;
   2) поява конкурентів;
   3) втрата одного або кількох найбільших користувачів.

У таблиці 4.10 відображено альтернативи впровадження стартап-проекту.

Таблиця 4.10 – Альтернативи ринкового впровадження стартап-проекту

№	Альтернатива (орієнтований комплекс заходів) ринкової поведінки	Ймовірність отримання ресурсів	Строки реалізації
1	Розробка модуля для популярних СУБД, веб-сайту та пошукової системи.	0.5	1 рік
2	Розробка модуля для популярних СУБД, веб-сайту, клієнтського API для популярних мов програмування та пошукової системи.	0.7	1.5 роки
3	Розробка розширення для Postgres та MySQL у вигляді динамічної бібліотеки, що об’єднує базу та пошукову системи.	0.7	1.5 роки

4.5 Розробка ринкової стратегії проекту

У таблицях 4.11 та 4.12 відображено вибір цільових груп користувачів та альтернативи розвитку продукту відповідно.

Таблиця 4.11 – Вибір цільових груп потенційних споживачів

№	Опис профілю цільової групи потенційних клієнтів	Готовність споживачів сприйти продукт	Орієнтовний помит цільової групи (сегменти)	Інтенсивність конкуренції в сегменті	Простота входу у сегмент
1	Веб-сайти середнього розміру, що потребують якісного пошуку	Потребують простих засобів для інтеграції	Попит серед веб-сайтів наступних типів: Інтернет-Магазини, інформаційні портали, каталоги продукції	Конкуренція із відкритими безкоштовними рішеннями	Проста, проте потребує реклами
2	Крупні веб-сайти, що потребують якісного пошуку	Потребують простих ефективних інтерфейсів для синхронізації даних, потребують підтримки шардування	Попит серед веб-сайтів наступних типів: Інтернет-Магазини, інформаційні портали, каталоги продукції	Конкуренція із відкритими безкоштовними рішеннями	Асоційована із витратами на спеціалізовані продажі та на рекламу.

Таблиця 4.12 – Альтернативи розвитку прокту

№	Обрана альтернатива розвитку проекту	Стратегія розвитку ринку	Ключові конкуретноспроможні позиції відповідно до обраної альтернативи	Базова стратегія розвитку
1	Набір критичної маси функціоналу, необхідної для виходу на ринок	На початкових етапах охоплювати ринок за рахунок користувачів CMS, надалі - розширюватись на основі ідеального профілю замовників	Наявність власної системи інтелектуального текстового пошуку	Різні тарифні плани націлені на різні групи користувачів, найбільш дешевий варіант повинен продаватися за собівартістю, оскільки його мета - змусити аудиторію користуватися сервіом.
2	Розвиток API та підтримка нових видів інтеграції сервісу	Створення розширення для популярних СУБД для серврної частини та JavaScript API - для клієнтської.	Проста інтеграції для будь-якої популярної платформи	Додаткова плата за шардування даних.

Для того, аби вести конкурентну боротьбу на обрано ринку, необхідно мати стратегії конкурентної поведінки, які відображено у таблиці 4.13.

Таблиця 4.13 – Визначення базової стратегії конкурентної поведінки

№	Чи є проект першопрохідцем на ринку	Чи буде компанія шукати нових споживачів чи забирати існуючих у конкурентів	Чи буде компанія копіювати основні характеристики товару конкурента і які?	Стратегія конкурентної поведінки
1	Проект є одним із першопрохідців на ринку	Компанія будет як шукати нових споживачів, так і переносити існуючі рішення з ElasticSearch на реалізовану платформу	Компанія будет оглядатися на існуючі відкриті рішення, такі як Elastic Search та Apache Solk	Компанія буде пропонуватиме готовий сервіс, який конкурує за рахунок якості пошуку та швидкодії

У таблиці 4.14 розкрито стратегії позиціонування продукту.

Таблиця 4.14 – Визначення стратегії позиціонування

#	Вимоги до товару цільової аудиторії	Базова стратегія розвитку	Ключові конкурентоспроможні позиції власного стартап-проекту	Вибір асоціацій, які можні мають сформувати комплексну позицію власного проекту
1	Швидкість обробки запитів	Розробка виконується із використанням C++ для максимально ефективного використання апартних ресурсів	Досвід використання SIMD інструкцій	а) гнучка мова для опису структури; б) гнучка мова генерування запитів; в) ефективний протокол взаємодії клієта із сервером.
2	Висока якість пошуку	Застосовуються технологій машинного навчання для ефектифного застосування факторів ранжування	Досвід команди у застосуванні ансамблів дерев та нейронних мереж	а) модель векторного представлення нграм, слів та фраз; б) врахування факторів ранжування на основі ансамблів дерев.

4.5.1 Розробка маркетингової програми стартап-проекту

У таблиці 4.15 сформульовано ключові переваги концепції потенційного товару.

Таблиця 4.15 – Визначення ключових переваг концепції потенційного товару

#	Потреба	Вигода, яку пропонує товар	Ключові переваги перд конкурентами (існуючі або такі, що потрібно створти)
1	Необхідність сервісної архітектури пошукової системи	Якісний пошук, що маштабується та не потребує ресурсів на підтримку	Алгоритми машинного навчання
2	Необхідність швидкого простого доступу до функцій	Наявність пакетів для швидкої інтеграції із застосуванням найпопулярніших мов	Наявність і серверного, і клієнтського варіанту інтеграції із сервісом

У таблиці 4.16 зображено визначення меж встановлення ціни.

Таблиця 4.16 – Визначення меж встановлення ціни

#	Рівень цін на товари-замінники	Рівень цін на товари-аналоги	Рівень цільової групи споживачів	Верхня та нижня меже встановлення ціни на товар

У таблиці 4.17 відображено принципи формування системи збуту.

Таблиця 4.17 – Формування системи збуту

#	Специфіка закупівельної поведінки цільових клієнтів	Функції збуту, які має виконувати постачальнику товару	Глибина каналу збуту	Оптимальна система збуту
1	Довгострокова підписка із збільшенням кількості шардів та реплік	Консультувати із застосування вискокорівневого API	2	Збут через провайдерів хмарних обчислень та систем розгортання приватних сервісів

У таблиці 4.18 відображено концепції маркетингових комунікацій.

Таблиця 4.18 – Концепції маркетингових комунікацій

#	Специфіка поведінки цільових клієнтів	Функції збуту, якими користуються цільові клієнти	Ключові позиції, обрані для позиціонування	Завдання рекламного повідомлення	Концепція рекламного звернення
1	Цільові клієнти шукають відкриті продукти та сервіси, що надають високу якість пошуку	Керування тарифними планами	Сервіси малого розміру 2гб оперативної пам’яті, 8гб, 16гб, 32гб, 64гб	Зареєструвати користувача	Проста і швидка інтеграція із сервісом
2	Ефективна обробка значних об’ємів даних в режимі онлайн	Надання інструкцій для розгортання	Реплікація даних та синхронізація між репліками	Зареєструвати користувача	Максимізація кількості запитів, оброблених за одиницю часу
3	Обробка значних об’ємів даних	Консультування із маштабування	Шарди різних розмірів	Зареєструвати користувача	Ефективне маштабування

Висновки до розділу

Дослідження в даній магістерській дисертації мають перспективи зростання до повноцінної компанії за рахунок поступового охоплення ринку інформаційного пошуку, витісняючи відкриті рішення, побудовані на базі Apache Lucene, оскільки витрати на обслуговування серверів із таким програмним забезпеченням є більш витратним, а також при цьому демонструє нижчу якість та швидкість пошуку.

ВИСНОВКИ ПО РОБОТІ ТА ПЕРСПЕКТИВИ ПОДАЛЬШИХ ДОСЛІДЖЕНЬ

В ході виконання магістерської дисертації вдалося розробити систему мультимодального текстового пошуку, що характеризується:

а) можливістю здійснювати пошук по документах завчасно невідомої структури;
б) варіативністю пошукової видачі релевантних документів;
в) врахуванням семантики запиту при генеруванні пошукової видачі;
г) можливістю оброляти тисячі запитів за 1 секунду.

В роботі поєднано використання ймовірнісних методів текстового пошуку із моделями машинного навчання на основі дерев рішень та їх ансамблів:

а) CART;
б) лісів рішень;
в) градієнтного бустингу.

В результаті дослідження можна зробити висновки, що якість текстового пошуку значною мірою залежить від якості розширення ключових слів пошукових запитів. При цьому застосування даної техніки сповільнює роботу системи і ставить високі умови до вибору та правильної реалізації швидкого пошуку у багатовимірному розрідженому просторі.

Подальшими дослідженнями за даною темою можуть стати:

- покращення векторної моделі слів та векторного представлення алгоритмів;
- реалізація діалекту SQL для роботи з даними;
- реалізувати можливість розширення функціоналу систему за рахунок інтеграції користувацьких плагінів;
- індивідальна адаптація моделі;
- вдосконалення та адаптація алгоритмів машинного навчання для даної задачі.

ПЕРЕЛІК ПОСИЛАНЬ

1. Згуровський М. З. Основи системного аналізу / М. З. Згуровський, Н. Д. Панкратова. — 1 вид. — Видавнича група BHV, 2007. — 546 с.
2. Mikolov, Tomas. Distributed Representations of Words and Phrases and their Compositionality / Tomas Mikolov, Ilya Sutskever, Jeffrey Dean // CoRR. — 2013. — Vol. abs/1310.4546, no. 7. [Електронний ресурс]. — Режим доступу: http://arxiv.org/abs/1310.4546.
3. Harris, Zellig S. Distributional Structure / Zellig S. Harris // WORD. — 1954. — Vol. 10, no. 2. — Pp. 54–60.
4. Anderson, J. A. Neurocomputing: Foundations of Research / J. A. Anderson, E. Rosenfeld. — Cambridge, MA, USA: MIT Press, 1988. — 101 pp.
5. Baroni, Marco. Distributional Memory: A General Framework for Corpus-based Semantics / Marco Baroni, Alessandro Lenci // Comput. Linguist. — 2010. — dec. — Vol. 36, no. 4. — Pp. 673–721. [Електронний ресурс]. — Режим доступу: http://dx.doi.org/10.1162/coli_a_00016.
6. Mikolov, Tomas. Distributed Representations of Words and Phrases and their Compositionality / Tomas Mikolov, Ilya Sutskever, Jeffrey Dean // CoRR. — 2013. — Vol. abs/1310.4546. [Електронний ресурс]. — Режим доступу: http://arxiv.org/abs/1310.4546.
7. Indyk, Piotr. Approximate nearest neighbors: towards removing the curse of dimensionality / Piotr Indyk, Rajeev Motwani // ACM Symposium on the Theory of Computing. — 1998. — Vol. 1, no. 2. — Pp. 604–613.
8. Similarity search in high dimensions via hashing. / Aristides Gionis, Piotr Indyk, , Rajeev Motwani // Proceedings of the 25th International Conference on Very Large Data Bases. — 1999. — Vol. 1, no. 2. — Pp. 10–15.
9. Multi-probe LSH: Efficient Indexing for High-dimensional Similarity Search / Qin Lv, William Josephson, Zhe Wang et al. // Proceedings of the 33rd International Conference on Very Large Data Bases. — Vol. 1 of VLDB '07. — VLDB Endowment, 2007. — Pp. 950–961. [Електронний ресурс]. — Режим доступу: http://dl.acm.org/citation.cfm?id=1325851.1325958.

10. Chierichetti, Flavio. LSH-Preserving Functions and Their Applications / Flavio Chierichetti, Ravi Kumar // J. ACM. — 2015. — nov. — Vol. 62, no. 3. — Pp. 33:1–33:25. [Електронний ресурс]. — Режим доступу: http://doi.acm.org/10.1145/2816813.
11. Bentley, Jon Louis. Multidimensional Binary Search Trees Used for Associative Searching / Jon Louis Bentley // Commun. ACM. — 1975. — sep. — Vol. 18, no. 9. — Pp. 509–517. [Електронний ресурс]. — Режим доступу: http://doi.acm.org/10.1145/361002.361007.
12. Kamel, Ibrahim. On Packing R-trees / Ibrahim Kamel, Christos Faloutsos // Proceedings of the Second International Conference on Information and Knowledge Management. — CIKM '93. — New York, NY, USA: ACM, 1993. — Pp. 490–499. [Електронний ресурс]. — Режим доступу: http://doi.acm.org/10.1145/170088.170403.
13. Shi, Zhendong. An Empirical Study of BM25 and BM25F Based Feature Location Techniques / Zhendong Shi, Jacky Keung, Qinbao Song // Proceedings of the International Workshop on Innovative Software Development Methodologies and Practices. — InnoSWDev 2014. — New York, NY, USA: ACM, 2014. — Pp. 106–114. [Електронний ресурс]. — Режим доступу: http://doi.acm.org/10.1145/2666581.2666594.
14. Stepanov, Alexander. Elements of Programming / Alexander Stepanov, Paul McJones. — 1st edition. — Addison-Wesley Professional, 2009. — 274 pp.
15. Cormen, Thomas H. Introduction to Algorithms, Third Edition / Thomas H. Cormen, Charles E. Leiserson, Ronald L. Rivest. — 3rd edition. — The MIT Press, 2009. — 1100 pp.
16. Austern, Matt. Untangling the Balancing and Searching of Balanced Binary Search Trees / Matt Austern, Bjarne Stroustrup, Mikkel Thorup // Software - Practice & Experience. — 2003. — Vol. 13, no. 7. — Pp. 1–10.
17. University of California, Berkeley. Latency Numbers Every Programmer Should Know. — 2017. [Електронний ресурс]. — Режим доступу: https://people.eecs.berkeley.edu/~rcs/research/interactive_latency.html.

18. Bengio, Yoshua. A Neural Probabilistic Language Model / Yoshua Bengio, Ducharme, Pascal Vincent // J. Mach. Learn. Res. — 2003. — mar. — Vol. 3, no. 1. — Pp. 1137–1155. [Електронний ресурс]. — Режим доступу: http://dl.acm.org/citation.cfm?id=944919.944966.
19. Stepanov, Alexander A. From Mathematics to Generic Programming / Alexander A. Stepanov, Daniel E. Rose. — 1st edition. — Addison-Wesley Professional, 2014. — 416 pp.
20. Enriching Word Vectors with Subword Information / Piotr Bojanowski, Edouard Grave, Armand Joulin, Tomas Mikolov // arXiv. — 2016. — Vol. 13, no. 7.
21. Baeza-Yates, Ricardo. Modern Information Retrieval / Ricardo Baeza-Yates, Berthier Ribeiro-Neto. — Harlow: Addison Wesley, 1999. — P. 430.
22. Rocchio, J. J. Relevance feedback in information retrieval / J. J. Rocchio. — Harlow: Addison Wesley, 1971. — Pp. 313–323.
23. RCV1: A New Benchmark Collection for Text Categorization Research / David D. Lewis, Yiming Yang, Tony G. Rose, Fan Li // JMLR. — 2004. — Vol. 5. — Pp. 361–397.
24. Jurafsky, Dan. Speech and Language Processing: An Introduction to Natural Language Processing, Computational Linguistics and Speech Recognition / Dan Jurafsky, James H. Martin. — 2nd edition. — Englewood Cliffs, NJ: Prentice Hall, 2008. — P. 900.
25. Theobald, Martin. TopX: Efficient and versatile top-*k* query processing for semistructured data / Martin Theobald, Ralf Schenkel, Gerhard Weikum // VLDB Journal. — 2008. — Vol. 17, no. 1. — Pp. 81–115.
26. Dice, L. R. Measures of the amount of ecologic association between species / L. R. Dice // Journal of Ecology. — 1945. — Vol. 26, no. 1. — Pp. 297–302.
27. Domingos, Pedro. A Unified Bias-Variance Decomposition for Zero-One and Squared Loss / Pedro Domingos // Proc. National Conference on Artificial Intelligence and Proc. Conference Innovative Applications of Artificial Intelligence. — AAAI Press / The MIT Press, 2000. — Pp. 564–569.
28. Aberer, Karl. P-Grid: A Self-Organizing Access Structure for P2P Information Systems / Karl Aberer // Proc. International Conference on Cooperative Information Systems. — London, UK: Springer, 2001. — Pp. 179–194.

29. Callan, Jamie. Advances in information retrieval / Jamie Callan. — Recent Research from the Center for Intelligent Information Retrieval, 2000. — Pp. 127–150.
30. Carreras, Xavier. Boosting Trees for Clause Splitting / Xavier Carreras, Lluís Màrquez // Proceedings of the 2001 Workshop on Computational Natural Language Learning - Volume 7. — ConLL '01. — Stroudsburg, PA, USA: Association for Computational Linguistics, 2001. — Pp. 26:1–26:3. [Електронний ресурс]. — Режим доступу: https://doi.org/10.3115/1117822.1117839.
31. R., Stephen. Simple BM25 extension to multiple weighted fields / Stephen R., Hugo Z., Michael T. // Proc. CIKM. — 2004. — Pp. 42–49.
32. Smeulders, Arnold W. M. Content-Based Image Retrieval at the End of the Early Years / Arnold W. M. Smeulders, Marcel Worring, Simone Santin // IEEE Trans. Pattern Anal. Mach. Intell. — 2000. — Vol. 22, no. 12. — Pp. 1349–1380.
33. Oard, Douglas W. A survey of multilingual text retrieval: Tech. Rep. UMIACS-TR-96-19 / Douglas W. Oard, Bonnie J. Dorr. — College Park, MD, USA: Institute for Advanced Computer Studies, University of Maryland, 1996.
34. Ingwersen, Peter. The Turn: Integration of Information Seeking and Retrieval in Context / Peter Ingwersen, Kalervo Järvelin. — Secaucus, NJ, USA: Springer, 2005. — 343 pp.
35. Meadow, Charles T. Text Information Retrieval Systems / Charles T. Meadow, Donald H. Kraft, Bert R. Boyce. — Orlando, FL, USA: Academic Press, 1999. — 401 pp.
36. Agrawal, Rakesh. Diversifying Search Results / Rakesh Agrawal, Sreenivas Gollapudi, Alan Halverson // Proceedings of the Second ACM International Conference on Web Search and Data Mining. — WSDM '09. — New York, NY, USA: ACM, 2009. — Pp. 5–14. [Електронний ресурс]. — Режим доступу: http://doi.acm.org/10.1145/1498759.1498766.
37. Carterette, Ben. Robust Test Collections for Retrieval Evaluation / Ben Carterette // Proceedings of the 30th Annual International ACM SIGIR Conference on Research and Development in Information Retrieval.

— SIGIR ’07. — New York, NY, USA: ACM, 2007. — Pp. 55–62. [Електронний ресурс]. — Режим доступу: http://doi.acm.org/10.1145/1277741.1277754.

38. Carterette, Ben. On Rank Correlation and the Distance Between Rankings / Ben Carterette // Proceedings of the 32Nd International ACM SIGIR Conference on Research and Development in Information Retrieval. — SIGIR ’09. — New York, NY, USA: ACM, 2009. — Pp. 436–443. [Електронний ресурс]. — Режим доступу: http://doi.acm.org/10.1145/1571941.1572017.
39. Collins-Thompson, Kevyn. Reducing the Risk of Query Expansion via Robust Constrained Optimization / Kevyn Collins-Thompson // Proceedings of the 18th ACM Conference on Information and Knowledge Management. — CIKM ’09. — New York, NY, USA: ACM, 2009. — Pp. 837–846. [Електронний ресурс]. — Режим доступу: http://doi.acm.org/10.1145/1645953.1646059.
40. P., Vries Arjen. Relevance Information: A Loss of Entropy but a Gain for IDF? / Vries Arjen P., Thomas Roelleke // Proceedings of the 28th Annual International ACM SIGIR Conference on Research and Development in Information Retrieval. — SIGIR ’05. — New York, NY, USA: ACM, 2005. — Pp. 282–289. [Електронний ресурс]. — Режим доступу: http://doi.acm.org/10.1145/1076034.1076084.

ДОДАТОК А. ІЛЮСТРАТИВНІ МАТЕРІАЛИ ДОПОВІДІ

ДОДАТОК Б. ЛІСТИНГ КОДУ

Ймовірнісний пошук документів

listings/common.h

```
#include <algorithm>
#include <sstream>
#include <fstream>
#include <chrono>
#include <cmath>
#include <iostream>
#include <cstring>
#include <vector>
#include <unordered_map>
#include <set>

namespace stl {
struct not_symbol {
  bool operator()(char x) const {
    return (x < 'a' || x > 'z') && x != '␣';
  }
};

void normalize_string(std::string& x) {
  std::for_each(std::begin(x), std::end(x), [](auto &x) { x = std::tolower(x); });
  x.erase(std::remove_if(std::begin(x), std::end(x), not_symbol{}), std::end(x));
}

template<typename ForwardIterator, typename OutputIterator>
std::pair<OutputIterator, size_t> copy_bounded(ForwardIterator first,
    ForwardIterator last, OutputIterator out, size_t n) {
  size_t i = 0;
  while (first != last && i < n) {
    *out = *first;
    ++out;
    ++first;
    ++i;
  }
  return {out, i};
}

template<typename ForwardIterator, typename OutputIterator, typename
    UnaryFunction>
```

```
std::pair<OutputIterator, size_t> transform_bounded(ForwardIterator first,
    ForwardIterator last, OutputIterator out, size_t n, UnaryFunction
    transformation) {
  size_t i = 0;
  while (first != last && i < n) {
    *out = transformation(*first);
    ++out;
    ++first;
    ++i;
  }
  return {out, i};
}

template<typename ForwardIterator, typename N>
N unique_count(ForwardIterator first, ForwardIterator last, N n) {
  if (first == last) return n;
  auto slow = first;
  ++n;
  ++first;
  while (first != last) {
    if (*slow != *first) {
      ++n;
      slow = first;
    }
    ++first;
  }
  return n;
}

template<typename T>
struct totally_ordered {
  friend
  inline
  bool operator>(const T& x, const T& y) {
    return (y < x);
  }

  friend
  inline
  bool operator>=(const T& x, const T& y) {
    return !(x < y);
  }

  friend
  inline
  bool operator<=(const T& x, const T& y) {
    return !(y < x);
```

```
  }
};


template<typename T>
struct word : totally_ordered<word<T>> {
  T data = 0;
  word() = default;
  word(const T& data) : data(data) {}
  word(const word& x) : data(x.data) {}

  word(const char* sequence, size_t n) {
    // memcpy(&data, sequence, std::min(sizeof(T), n));
    memcpy(&data, sequence, n);
  }

  friend
  inline
  bool operator==(const word& x, const word& y) {
    return x.data == y.data;
  }

  friend
  inline
  bool operator!=(const word& x, const word& y) {
    return !(x == y);
  }

  friend
  inline
  bool operator<(const word& x, const word& y) {
    return x.data < y.data;
  }

  struct hash {
    size_t operator()(const word& x) const {
      return std::hash<T>{}(x.data);
    }
  };
};

typedef word<uint32_t> word32_t;
typedef word<uint64_t> word64_t;

template<typename N, typename W>
struct posting_list {
  typedef N document_id_type;
```

```
    typedef W word_type;

    std::unordered_map<word_type, std::vector<document_id_type>, typename
      word_type::hash> data;

    void emplace(const document_id_type& document_id, const word_type& word) {
      data[word].emplace_back(document_id);
    }

    size_t documents_count(const word_type& word) {
      auto pos = data.find(word);
      if (pos == std::end(data)) {
        return 0;
      }
      return stl::unique_count(std::begin(pos->second), std::end(pos->second),
      size_t{0});
    }

    size_t term_frequency(const word_type& word, const document_id_type& doc_id)
      const {
      auto pos = data.find(word);
      if (pos == std::end(data)) {
        return 0;
      }
      auto pair = std::equal_range(std::begin(pos->second), std::end(pos->second),
      doc_id);
      return pair.second - pair.first;
    }

    template<typename OutputIterator>
    OutputIterator documents(const word_type& term, OutputIterator out) {
      auto& docs = data[term];
      return std::copy(std::begin(docs), std::end(docs), out);
    }

    template<typename OutputIterator, typename Transformation>
    OutputIterator documents(const word_type& term, OutputIterator out,
      Transformation transform) {
      auto& docs = data[term];
      return std::transform(std::begin(docs), std::end(docs), out, transform);
    }
  };

  template<typename PostingList>
  struct inverted_index {
    typedef PostingList posting_list_type;
    typedef typename posting_list_type::word_type word_type;
```

```
171   typedef typename posting_list_type::document_id_type document_id_type;
      typedef size_t size_type;

      posting_list_type posting_list;
      std::unordered_map<word_type, size_type, typename word_type::hash> frequencies;
176
      void emplace(document_id_type document_id, const word_type& term) {
        ++frequencies[term];
        posting_list.emplace(document_id, term);
      }
181
      template<typename ForwardIterator>
      void emplace(const document_id_type& document_id, ForwardIterator first,
        ForwardIterator last) {
        std::for_each(first, last, [this, document_id](const word_type& x) {
        emplace(document_id, x); });
      }
186
      template<typename OutputIterator, typename Transformation>
      OutputIterator documents(const word_type& term, OutputIterator out,
        Transformation transform) {
        return posting_list.documents(term, out, transform);
      }
191
      size_type count(const word_type& x) {
        auto pos = frequencies.find(x);
        if (pos == std::end(frequencies)) {
          return size_type{0};
196     }
        return pos->second;
      }

      size_type documents_count(const word_type& word) {
201     return posting_list.documents_count(word);
      }

      auto term_frequency(const word_type& word, const document_id_type& doc_id)
        const {
        return posting_list.term_frequency(word, doc_id);
206   }

    };

    template<typename InvertedIndex>
211 struct okapi_bm25 {
      typedef InvertedIndex inverted_index_type;
      typedef typename inverted_index_type::word_type word_type;
```

```
    typedef typename inverted_index_type::document_id_type document_id_type;

    inverted_index_type inverted_index;
    size_t documents_count = 0;

    template<typename ForwardIterator>
    void emplace(const document_id_type& document_id, ForwardIterator first,
      ForwardIterator last) {
      ++documents_count;
      inverted_index(document_id, first, last);
    }
  };

  template<typename ForwardIterator, typename ForwardIteratorOutput, typename
      Function>
  ForwardIteratorOutput cumulate(ForwardIterator first, ForwardIterator middle,
      ForwardIterator last, ForwardIteratorOutput output, Function f) {
    if (first == middle) return output;
    *output = f(first, middle);
    ++first;
    ++output;

    while (middle != last) {
      *output = f(first, middle);
      ++first;
      ++middle;
      ++output;
    }
    return output;
  }

  template<typename W = word32_t, typename S = size_t>
  struct search_ngram {
    typedef W word_type;
    typedef S size_type;
    typedef size_type document_id_type;

    typedef stl::posting_list<document_id_type, word_type> posting_list_type;
    typedef stl::inverted_index<posting_list_type> inverted_index_type;

    size_t terms_count = 0;


    inverted_index_type inverted_index;

    std::vector<std::string> documents;
    size_t n;
```

```
search_ngram() : n(1) {}
search_ngram(size_t n) : n(n) {}

virtual
std::vector<W> extract_terms(std::string document) {
  normalize_string(document);
  if (document.size() < n) {
    return std::vector<word_type>{};
  }
  const char* str = document.c_str();
  std::vector<word_type> terms(document.size() + 1 — n);
  stl::cumulate(str, str + n, str + document.length(), std::begin(terms),
  [this](const char *f, const char *l) { return word_type(f, n); });
  return terms;
}

void emplace(std::string document) {
  documents.emplace_back(document);
  if (document.size() < n) return;
  size_t document_id = documents.size();
  auto terms = extract_terms(document);
  terms_count += terms.size();
  inverted_index.emplace(document_id, std::begin(terms), std::end(terms));
}

void push_back(std::string document) {
  emplace(document);
}


template<typename OutputIterator>
OutputIterator find_k(std::string query, OutputIterator out, size_t k) {
  auto terms = extract_terms(query);
  std::vector<std::pair<document_id_type, float>> documents_scores;
  documents_scores.reserve(k * k * k);
  std::for_each(std::begin(terms), std::end(terms), [this,
  out=std::back_inserter(documents_scores)](const auto &term) mutable {
    inverted_index.documents(term, out, [](auto x) { return std::make_pair(x,
  0.0f); });
  });
  std::sort(std::begin(documents_scores), std::end(documents_scores));
  documents_scores.erase(std::unique(std::begin(documents_scores),
  std::end(documents_scores)), std::end(documents_scores));

  std::sort(std::begin(terms), std::end(terms));
  terms.erase(std::unique(std::begin(terms), std::end(terms)), std::end(terms));
```

```
301
    std::vector<float> idfs(terms.size());
    terms_idfs(std::begin(terms), std::end(terms), std::begin(idfs));
    std::for_each(std::begin(documents_scores), std::end(documents_scores),
    [this, &terms, &idfs](auto &x) {
      x.second = relevance(x.first, std::begin(terms), std::end(terms),
    std::begin(idfs));
306 });
    std::sort(std::begin(documents_scores), std::end(documents_scores), [](const
    auto& x, const auto& y) { return x.second > y.second; });
    return stl::transform_bounded(std::begin(documents_scores),
    std::end(documents_scores), out, k, [](const auto& x) {
      std::cout << x.second << std::endl;
      return x.first;
311 }).first;
  }

  template<typename ForwardIterator, typename OutputIterator>
  auto terms_idfs(ForwardIterator first, ForwardIterator last, OutputIterator
    output) {
316 size_t n = documents.size();
    return std::transform(first, last, output, [this, n](const auto& term) {
    return idf<float>(n, inverted_index.documents_count(term)); });
  }

  template<typename T>
321 T idf(T n, T nqi) {
    auto x = (n - nqi + T{0.5}) / (nqi + T{0.5});
    T r = std::log(x);
    return r;
  }
326

  template<typename ForwardIterator0, typename ForwardIterator1>
  float relevance(const document_id_type& doc_id, ForwardIterator0 first_terms,
    ForwardIterator0 last_terms, ForwardIterator1 idfs) const {
    float relevance = 0.0f;
331 float avgdl = float(terms_count) / float(documents.size());
    float k1 = 2.0f;
    float b = 0.75f;
    float D = documents[doc_id - 1].size() - n + 1;
    while (first_terms != last_terms) {
336   auto fqd = inverted_index.term_frequency(*first_terms, doc_id);
      relevance += *idfs * (fqd * (k1 + 1.0f)) / (fqd + k1 * (1 - b + b * D /
    avgdl));
      ++first_terms;
      ++idfs;
```

```
    }
    return relevance;
  }
};

template<typename Istream, typename SearchEngine>
void read_wikipedia_titles(Istream &cin, SearchEngine &search_engine, size_t n) {
  std::string str;
  for(size_t i = 0; std::getline(cin, str, '\t') && i < n; ++i) {
    search_engine.emplace(str);
    if (cin) {
      std::getline(cin, str, '\n');
    }
  }
}


struct stopwatch {
  std::chrono::time_point<std::chrono::high_resolution_clock> start_;

  stopwatch() {
    start();
  }

  void start() {
    start_ = now();
  }

  std::chrono::time_point<std::chrono::high_resolution_clock>  now() const {
    return std::chrono::high_resolution_clock::now();
  }

  double stop() {
    return std::chrono::duration_cast<std::chrono::nanoseconds>(now() -
    start_).count() / 1.0e9;
  }
};

template<typename W = word64_t, typename S = size_t>
struct search_words : search_ngram<W, S> {
  typedef W word_type;
  typedef S size_type;
  typedef size_type document_id_type;

  search_words(size_t n) {}
  virtual
```

```
386   std::vector<W> extract_terms(std::string document) {
        std::for_each(std::begin(document), std::end(document), [](auto& x) { x =
       std::tolower(x); });
        std::stringstream stream(document);
        std::vector<word_type> terms;
        std::string str;
391     while (stream >> str) {
          std::cout << str << std::endl;
          auto n = std::min(size_t{8},  str.length());
          terms.emplace_back(word_type{str.c_str(), n});
        }
396     return terms;
      }
    };
    }
```

Пошук по н-грамам

listings/e1_ngram.cpp

```
1   #include "common.h"

    int main(int argc, char* argv[]) {
      stl::search_words search_engine(3);
      search_engine.emplace("Why␣C++␣is␣better␣than␣Rust");
6     search_engine.emplace("Why␣Rust␣is␣better␣than␣C++");
      search_engine.emplace("Programming␣in␣C++");
      search_engine.emplace("Programming␣in␣Rust");
      search_engine.emplace("Good␣life");
      search_engine.emplace("Happy␣life␣life");
11    search_engine.emplace("Samsung␣8");

      stl::stopwatch stopwatch;
      auto end = std::chrono::high_resolution_clock::now();
      std::cout << "Created␣index␣in:␣" << stopwatch.stop() << "␣sec." << std::endl;
16
      stopwatch.start();
      std::vector<decltype(search_engine)::document_id_type> documents;
      const size_t k = 10;
      search_engine.find_k(argv[1], std::back_inserter(documents), k);
21    std::cout << "Retrieved␣" << documents.size() << "␣documents" << std::endl;

      std::for_each(std::begin(documents), std::end(documents), [&search_engine,
       i=0](const auto &x) mutable {
          std::cout << ++i << ".\t" << search_engine.documents[x - 1] << std::endl;
```

```
  });

  end = std::chrono::high_resolution_clock::now();
  std::cout << "Eveluated in: " << stopwatch.stop() << " sec." << std::endl;
}
```

Пошук по словах

listings/e2_term.cpp

```
#include "common.h"
#include <boost/tokenizer.hpp>

struct homedepot_item {
  size_t product_uid;
  std::string product_title;
  std::string search_term;
  double relevance;

  friend
  bool
  inline operator<(const homedepot_item& x, const homedepot_item& y) {
    return x.product_uid < y.product_uid;
  }

  struct less_product_uid {
    bool operator()(const homedepot_item& x, const homedepot_item& y) const {
      return x.product_uid < y.product_uid;
    }
  };

  struct less_product_title {
    bool operator()(const homedepot_item& x, const homedepot_item& y) const {
      return x.product_title < y.product_title;
    }
  };

  struct less_search_term {
    bool operator()(const homedepot_item& x, const homedepot_item& y) const {
      return x.search_term < y.search_term;
    }
  };
};

void tokenize(const std::string& x, homedepot_item& result) {
```

```
36    //"id","product_uid","product_title","search_term","relevance"
      boost::tokenizer<boost::escaped_list_separator<char>> tok(x);
      auto first = tok.begin();
      auto last = tok.end();
      if (std::distance(first, last) != 5) throw std::runtime_error("Invalid␣file␣
        format");
41
      ++first;
      result.product_uid = std::atoi(first->c_str());
      ++first;
      result.product_title = *first;
46    ++first;
      result.search_term= *first;
      ++first;
      result.relevance = std::atof(first->c_str());
    }
51
    void tokenize(std::ifstream& ifstream, homedepot_item &result, std::string&
        buffer) {
      std::getline(ifstream, buffer);
      tokenize(buffer, result);
    }
56
    template<typename Istream, typename OutputIterator, typename Function>
    OutputIterator transform_stream(Istream& cin, OutputIterator out, Function func) {
      while (cin) {
        *out = func(cin);
61      ++out;
      }
      return out;
    }

66  struct tokenize_homedepot_item {
      std::string* string_buffer;
      tokenize_homedepot_item() = default;
      tokenize_homedepot_item(std::string* x) : string_buffer(x) {}

71    template<typename Istream>
      homedepot_item operator()(Istream &cin) {
        homedepot_item x;
        tokenize(cin, x, *string_buffer);
        return x;
76    }
    };

    template<typename Istream, typename OutputIterator>
    OutputIterator copy_homedepot_item_from_stream(Istream& cin, OutputIterator out) {
```

```
  std::string buffer;
  return transform_stream(cin, out, tokenize_homedepot_item(&buffer));
}

template<typename OutputIterator>
OutputIterator copy_homedepot_item_from_stream(const char* str, OutputIterator
    out) {
  std::ifstream ifstream(str);
  std::string tmp;
  std::getline(ifstream, tmp);
  return copy_homedepot_item_from_stream(ifstream, out);
}

template<typename ForwardIterator, typename OutputIterator, typename
    BinaryPredicate, typename Transformation>
OutputIterator unique_transform(ForwardIterator first, ForwardIterator last,
    OutputIterator out, BinaryPredicate predicate, Transformation f) {
  if (first == last) {
    return out;
  }
  auto fast = first;
  ++fast;
  *out = f(*first);
  ++out;
  while (fast != last) {
    if (!predicate(*first, *fast)) {
      first = fast;
      *out = f(*first);
      ++out;
    }
    ++fast;
  }
  return out;
}

struct homedepot {
  std::vector<homedepot_item> homedepot_items;
  std::vector<std::string> product_titles;
  std::unordered_map<std::string_view, std::vector<homedepot_item*>> multimap;

  homedepot(const char* train_csv) {
    try {
      copy_homedepot_item_from_stream(train_csv,
    std::back_inserter(homedepot_items));
    } catch(...) {}
    std::sort(std::begin(homedepot_items), std::end(homedepot_items),
    homedepot_item::less_product_uid());
```

```
    unique_transform(std::begin(homedepot_items), std::end(homedepot_items),
    std::back_inserter(product_titles),
                     [](const auto& x, const auto &y) { return x.product_uid ==
    y.product_uid; },
                     [](const auto& x) { return x.product_title; });
126   for (auto& x : homedepot_items) {
        multimap[x.search_term].emplace_back(&x);
      }
    }
};
131

struct search_evaluation {
  template<typename SE, typename ForwardIterator>
  void operator()(SE &search_engine, ForwardIterator first, ForwardIterator last)
    {
136

  }
};

int main(int argc, char* argv[]) {
141  std::ios_base::sync_with_stdio(false);
  const char* train_csv =
    "/home/tshevchenko/Documents/Datasets/retrieval/homedepot/train.csv";
  homedepot train(train_csv);
  const size_t ngram_n = 3;
  stl::search_ngram search_engine(ngram_n);
146  std::for_each(std::begin(train.product_titles), std::end(train.product_titles),
    [&](const auto& x) { search_engine.emplace(x); });

  search_evaluation evaluation;
  for (const auto& x : train.multimap) {
    evaluation(search_engine, std::begin(x.second), std::end(x.second));
151  }


  stl::search_ngram search_engine(ngram_n);
  std::fstream cin(argv[1]);
156  if (!cin) {
    std::cout << "Invalid␣path";
    return -1;
  }

161  stl::stopwatch stopwatch;
  const size_t n = 100000;
  stl::read_wikipedia_titles(cin, search_engine, n);
  auto end = std::chrono::high_resolution_clock::now();
```

```
    std::cout << "Created␣index␣in:␣" << stopwatch.stop() << "␣sec." << std::endl;
166
    stopwatch.start();
    std::vector<decltype(search_engine)::document_id_type> documents;
    const size_t k = 10;
    search_engine.find_k(argv[1], std::back_inserter(documents), k);
171 std::cout << "Retrieved␣" << documents.size() << "␣documents" << std::endl;

    std::for_each(std::begin(documents), std::end(documents), [&search_engine,
     i=0](const auto &x) mutable {
        std::cout << ++i << ".\t" << search_engine.documents[x - 1] << std::endl;
    });
176
    end = std::chrono::high_resolution_clock::now();
    std::cout << "Eveluated␣in:␣" << stopwatch.stop() << "␣sec." << std::endl;

}
```